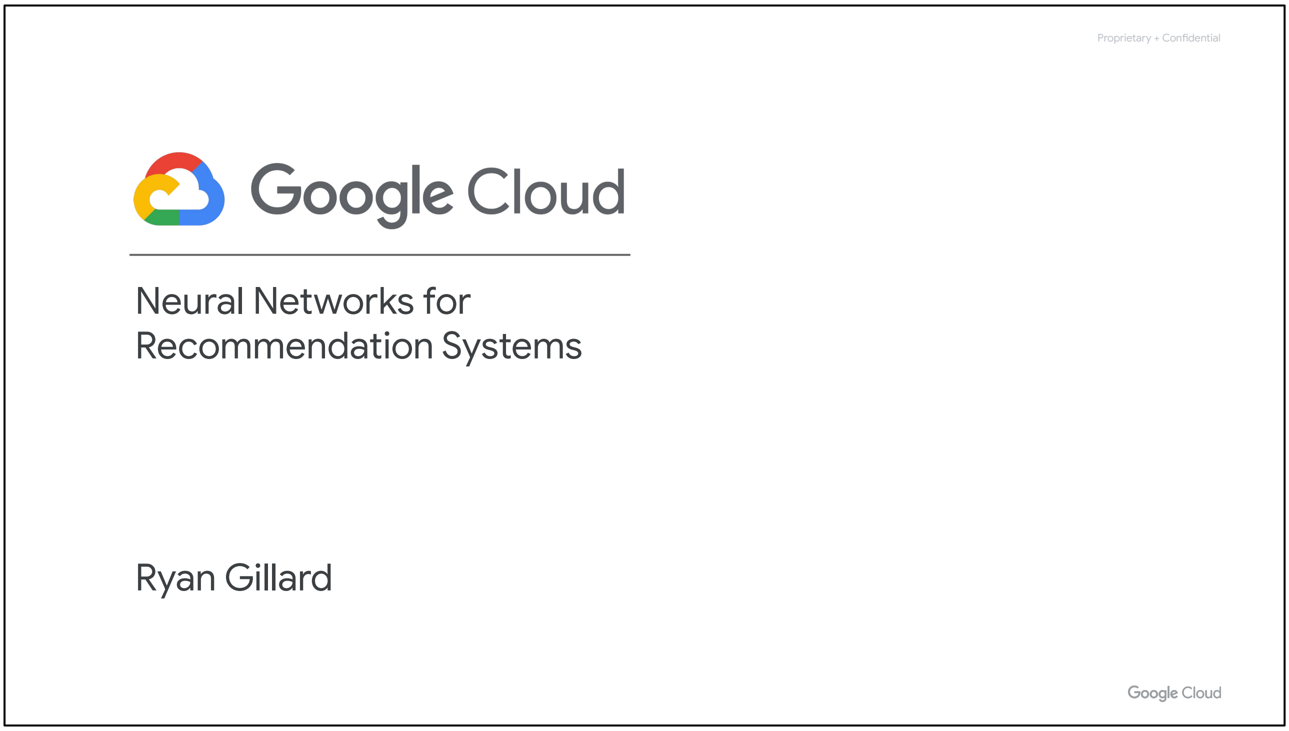
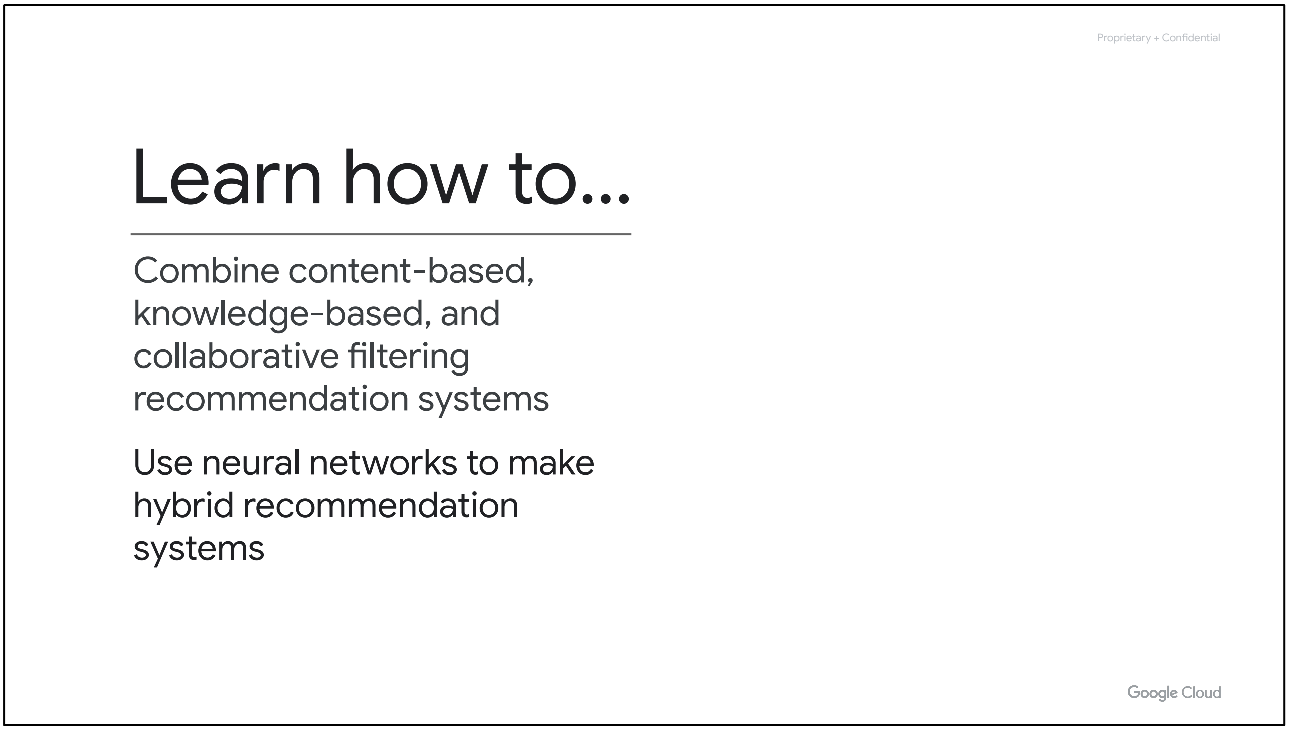


Google Cloud

Neural Networks for Recommendation Systems

Ryan Gillard

Hi I’m Ryan, I’m a machine learning scientist at Google, and I love applying math and machine learning to big data to better make sense of the world. In the previous modules, we learned how both content-based and collaborative filtering forms of recommender systems work. In this module, we will take advantage of the power of neural networks to create hybrid recommendation systems, using all that we have learned so far about recommendation systems put together.



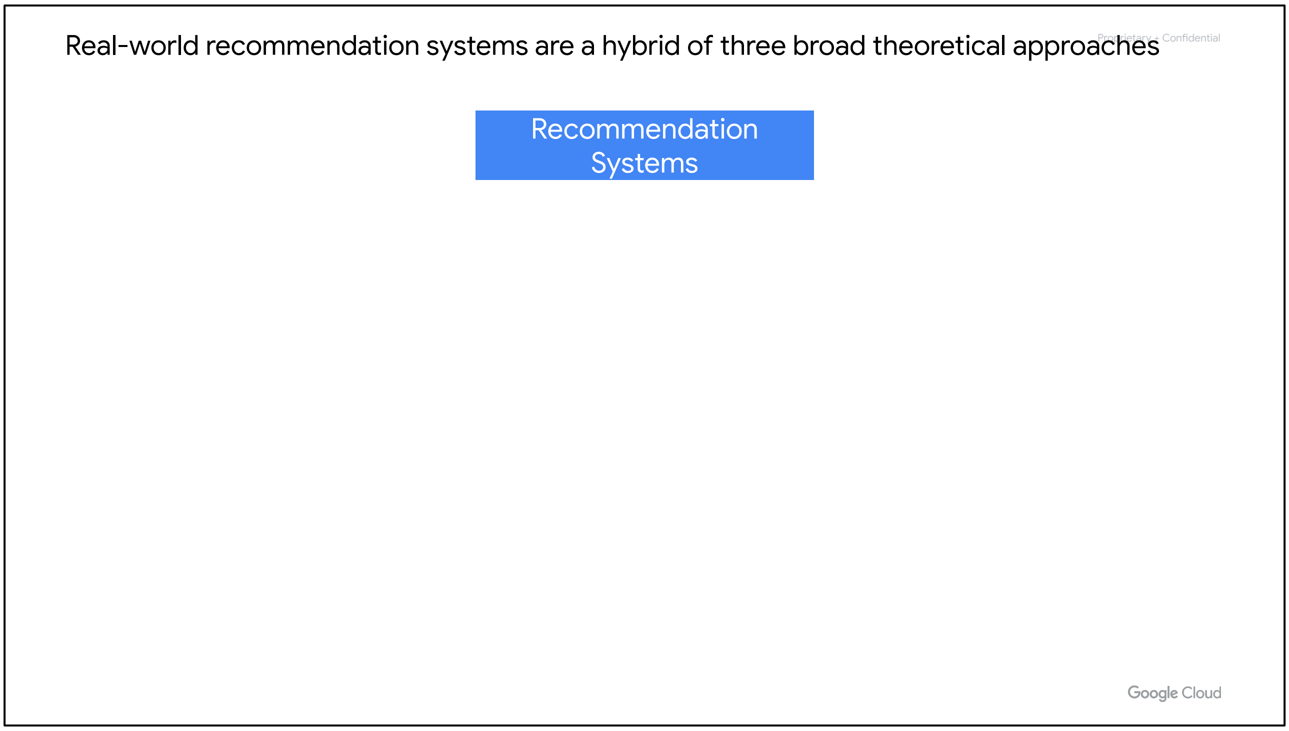
Learn how to...

Combine content-based, knowledge-based, and collaborative filtering recommendation systems

Use neural networks to make hybrid recommendation systems

Google Cloud

We'll learn how to do this by using neural networks to make hybrid recommendation systems.



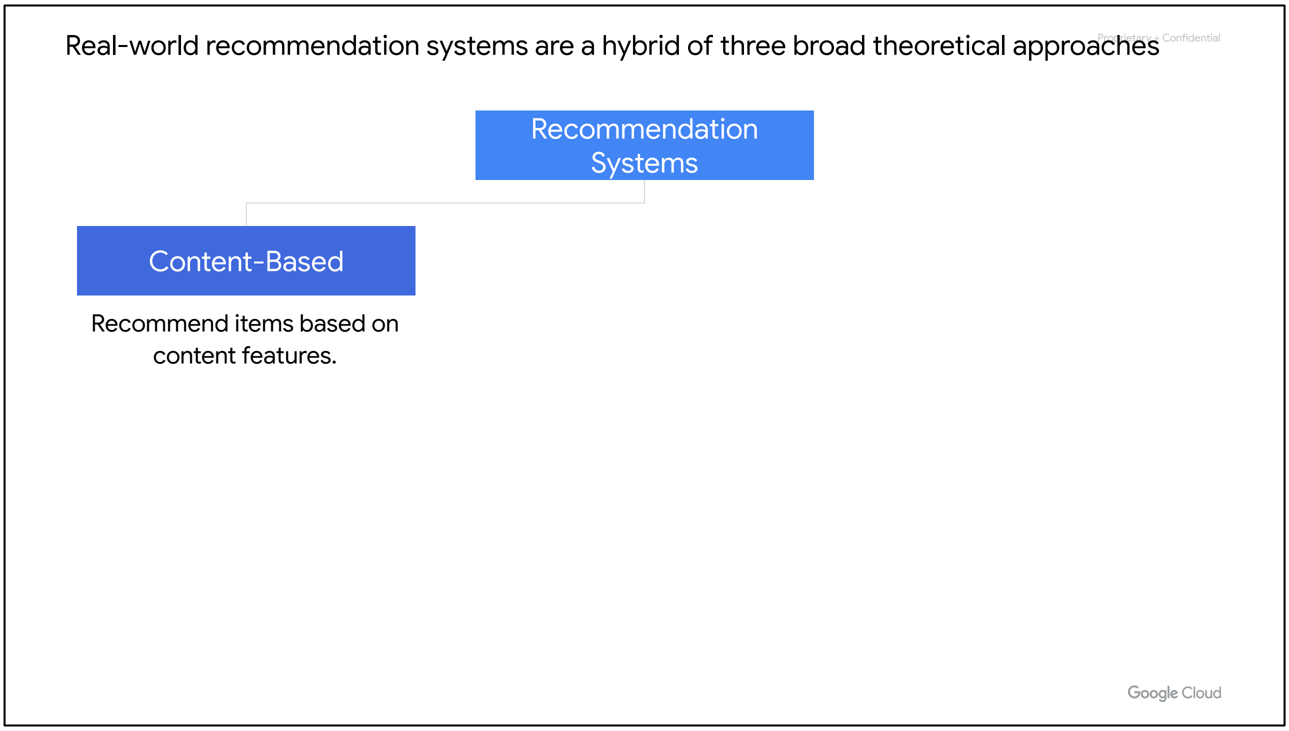
Proprietary + Confidential

Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems

Google Cloud

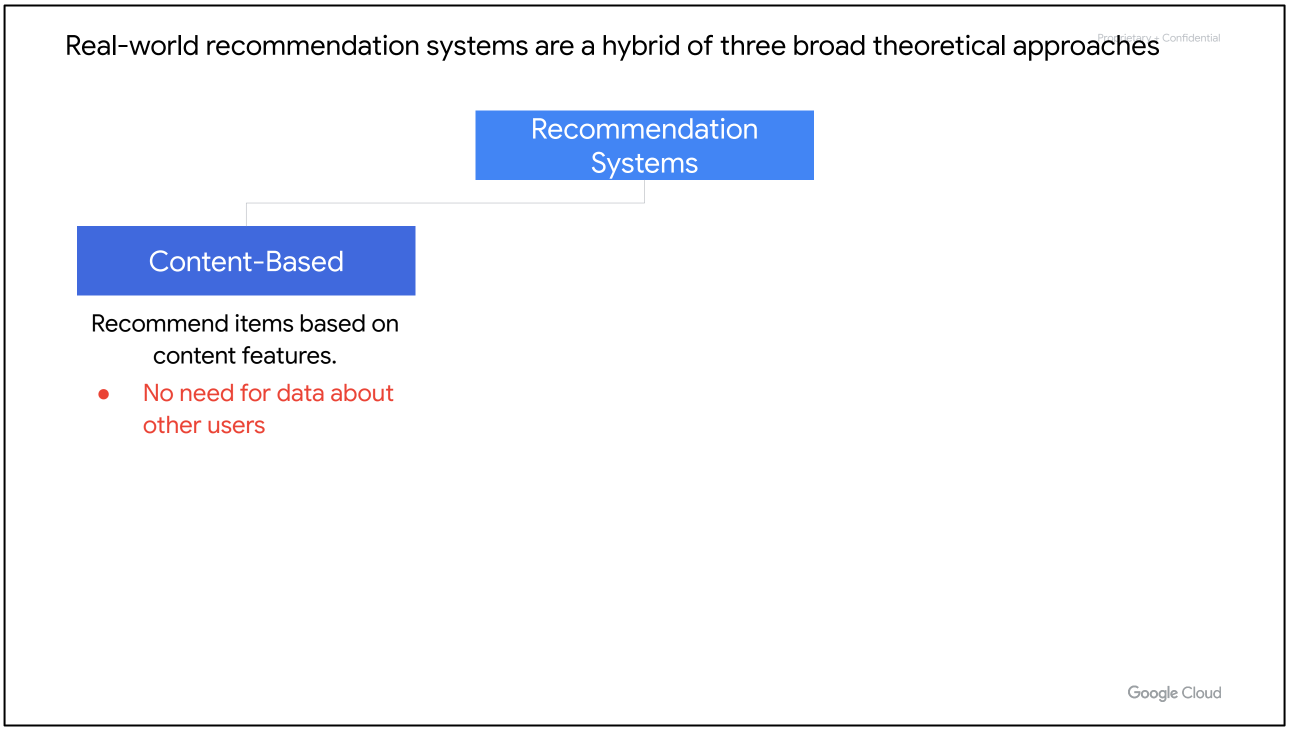
We've already learned several types of recommendation systems; however we used each in a vacuum, taking advantage of different types of data to try to make the best recommendations.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Proprietary + Confidential
Recommendation Systems
Content-Based
Recommend items based on content features.
Google Cloud

We saw in module 2 how to build content-based recommendation systems. This involved taking properties of the items. This could have been structured data such as the genre of movies, as seen in our earlier examples. It also could have been embeddings of the text description, images, or even audio and/or video preprocessed through sequence models. We take this matrix of item properties and multiply it with a user vector to get that user's representation in the item embedding space. We then can use one of multiple similarity measures to recommend similar items that the user would like.



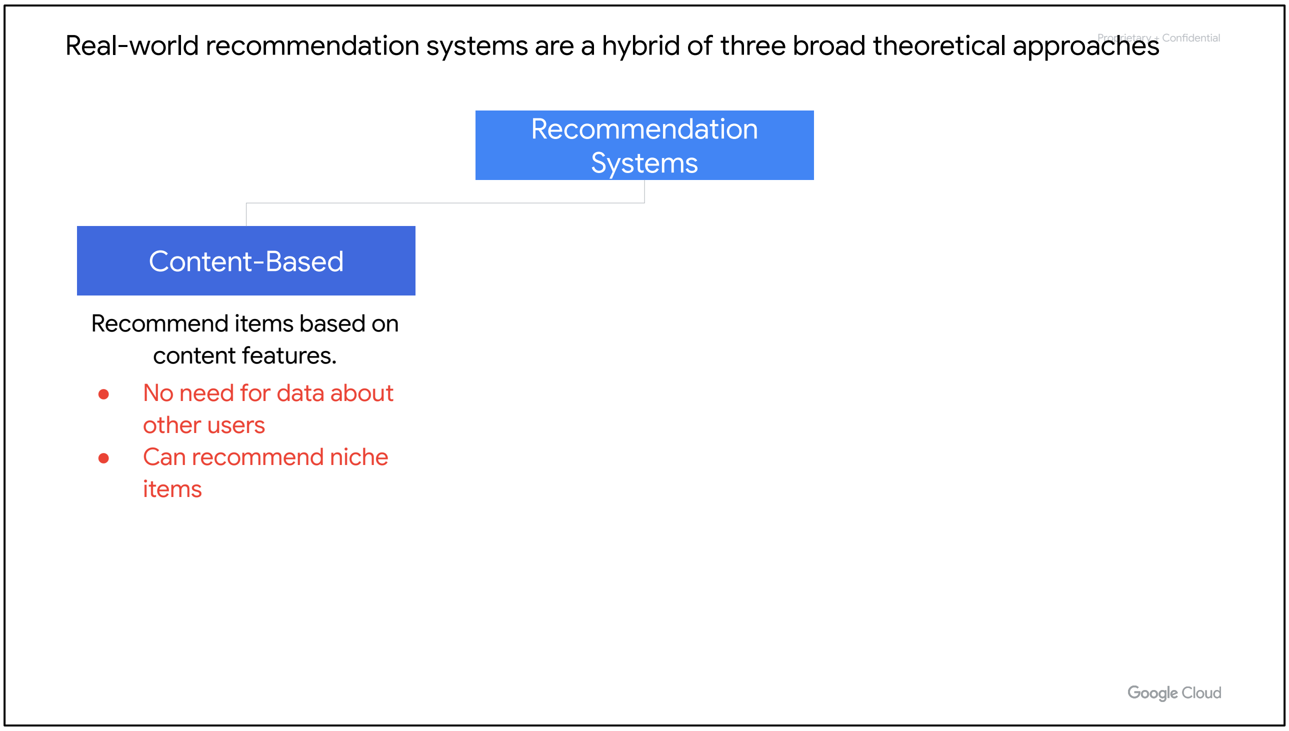
Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based

Recommend items based on content features.

- No need for data about other users

Content-based recommendation systems have many pros. First, there is no need for data about other users. We just need the data about the user of interest, and then we can use that to recommend similar items from the learned embedding space.



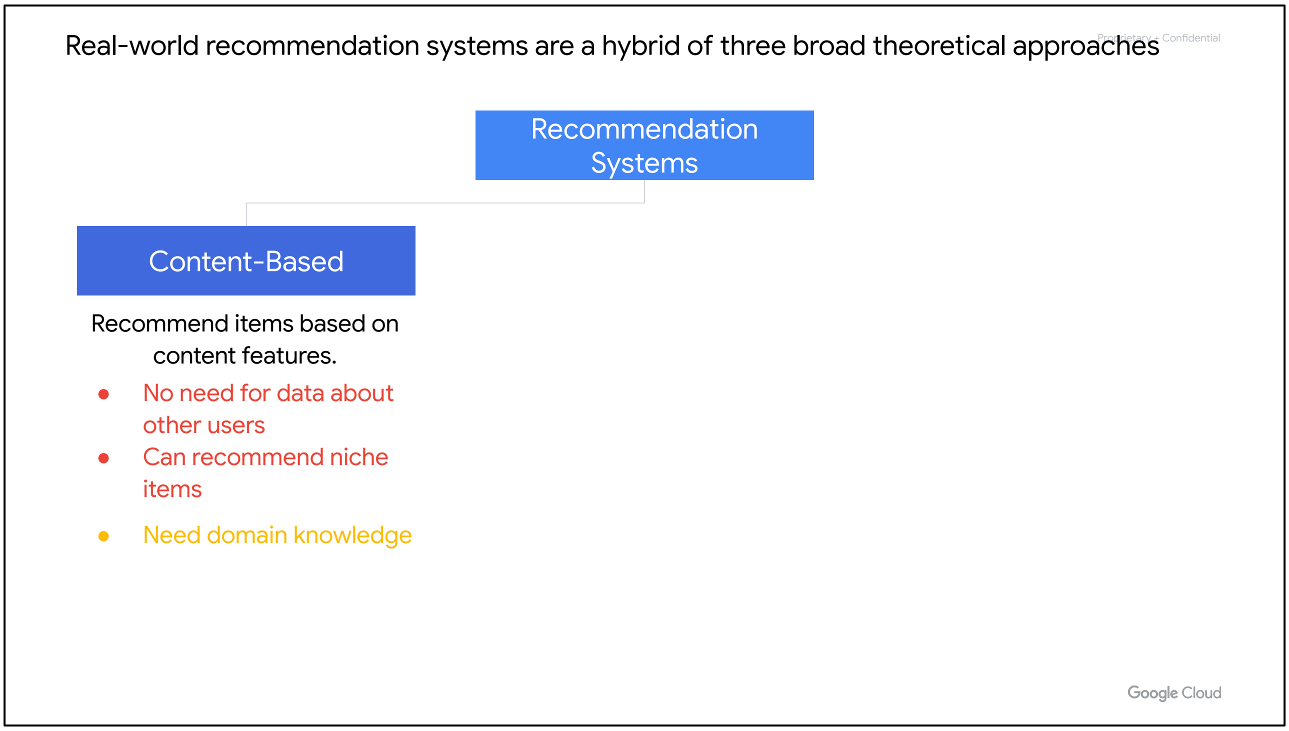
Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

Also, we can recommend niche items because we know about our users' specific tastes, which might not be shared amongst other users, and therefore we can recommend to meet those unique interests.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

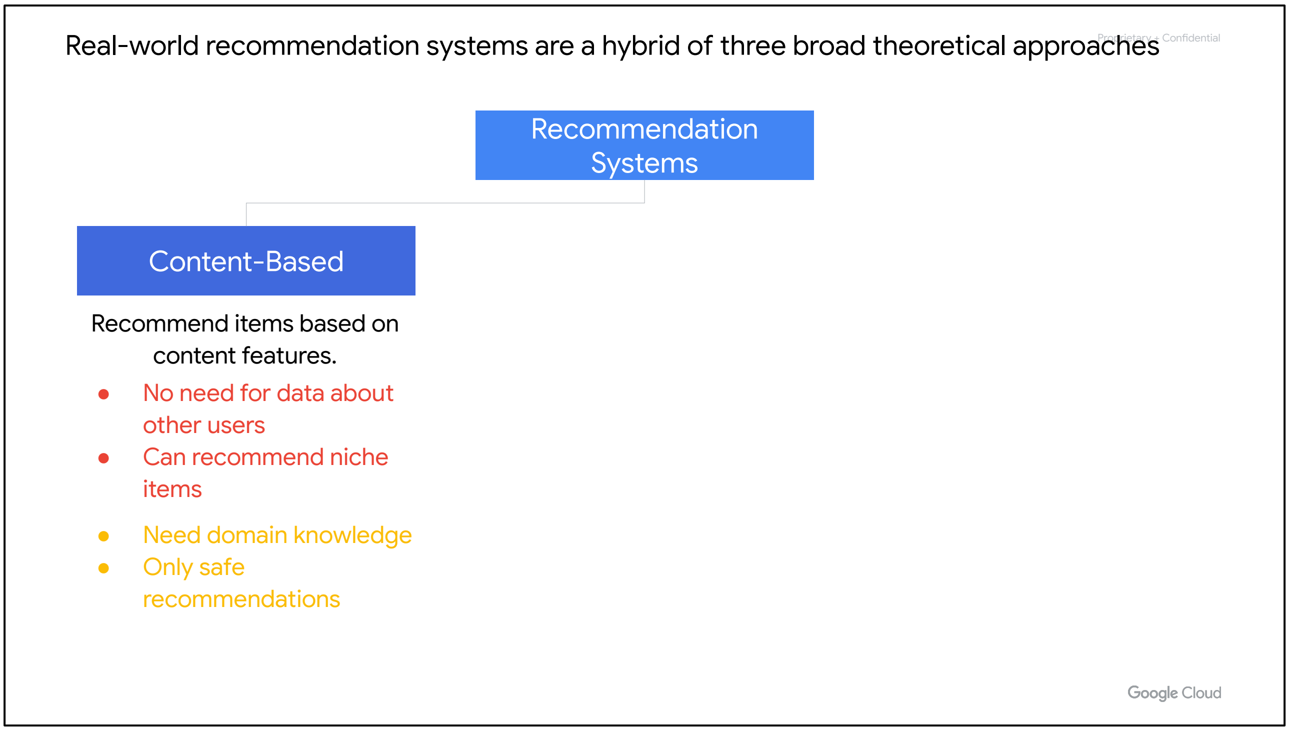
Recommendation Systems

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items
- Need domain knowledge

However, content-based recommendation systems have their cons. There needs to be domain knowledge. A human has to label the movie genres. A human has to enter the text description of the items. A human has to post the item picture. A human has to create the audio or video clip attached to the item. As you can see, there is a lot of involvement by expert humans who know their items very well.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

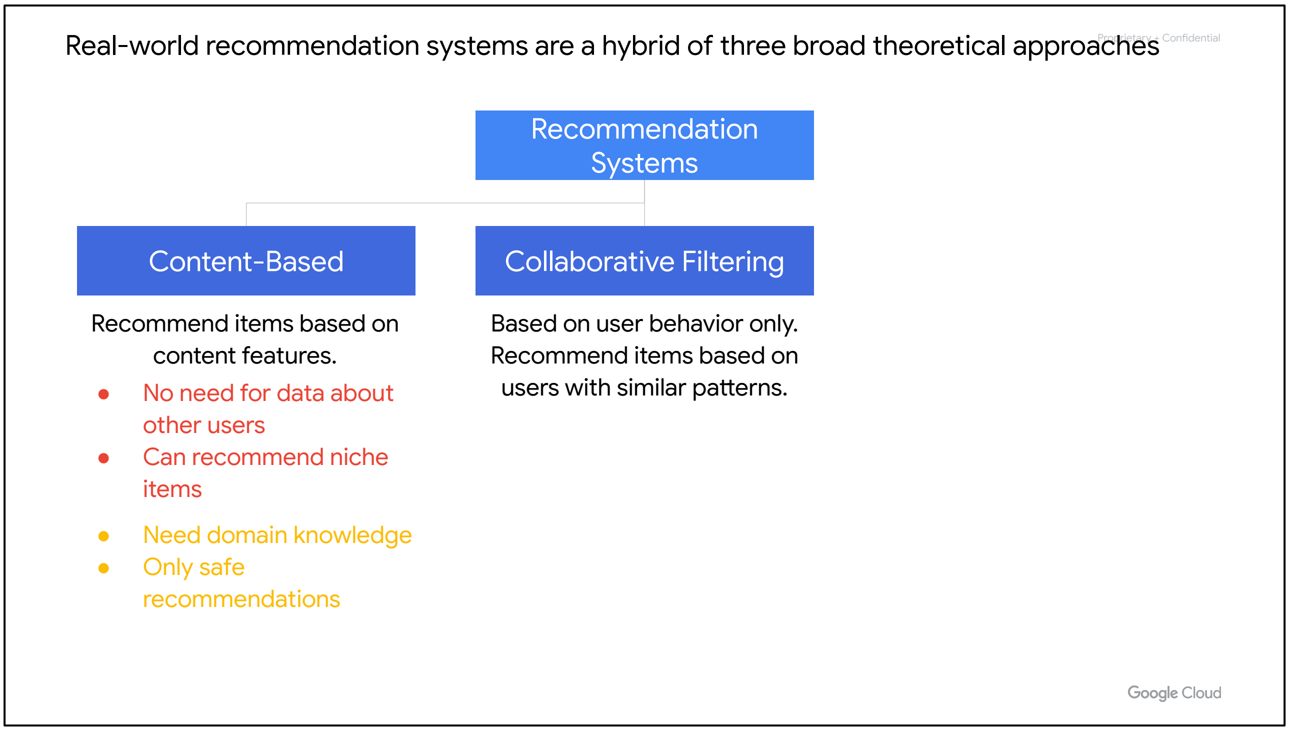
Recommendation Systems
Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Also, content-based recommendation systems tend to make only safe recommendations. They stay within a user's safe bubble of the embedding space. If a user has never expanded this bubble within that space, content-based recommendation systems will only recommend similar things, which will invariably end up within the bubble or near the edge. There isn't any information in a purely content-based recommendation system that can push a user outside their usual boundaries, to explore items that they didn't know they actually might like.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering

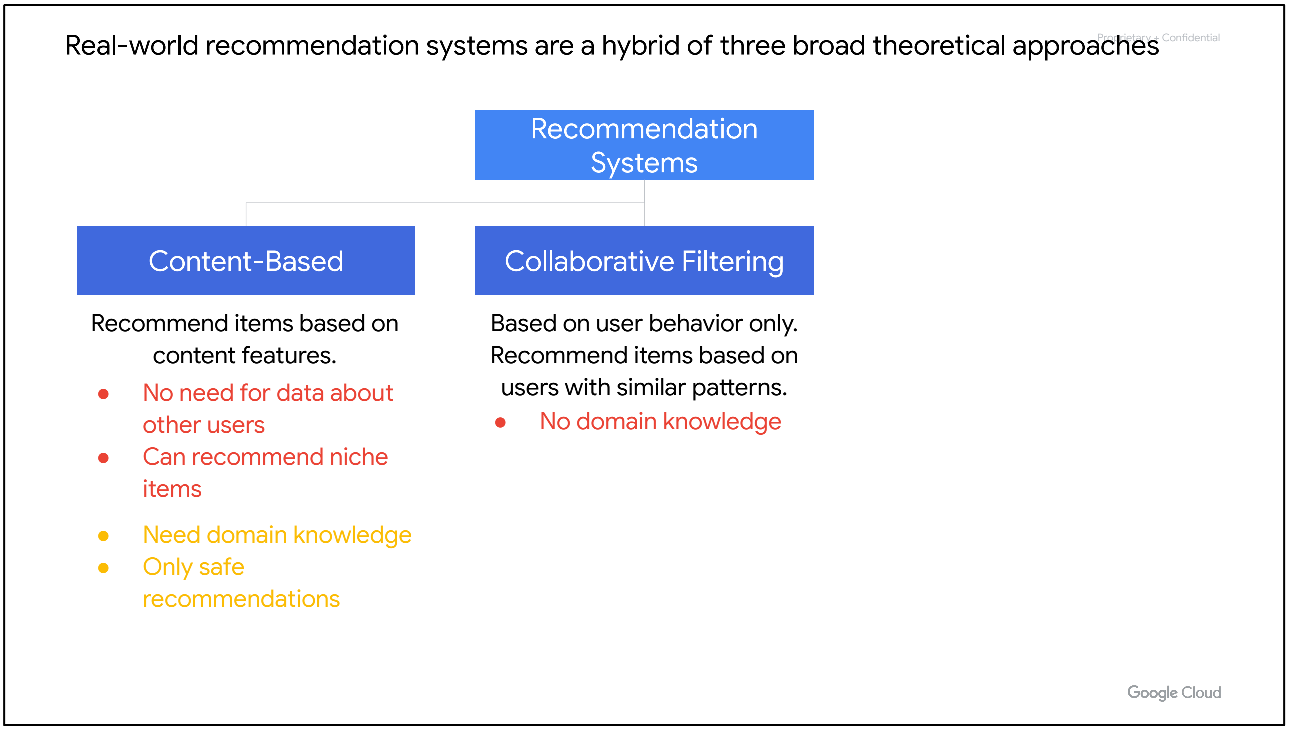
Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Based on user behavior only. Recommend items based on users with similar patterns.

We learned in module 3 how we can use WALSMatrixFactorization for collaborative filtering, to make recommendations based on user-item interactions. It essentially takes these interactions and learns the latent factors within them to best generalize these interactions. These latent factors create a d-dimensional embedding space where a user and item embedding are solved for simultaneously. Not only can we recommend items for users, but we can target users for items because we have the two embeddings.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering

Recommend items based on content features.

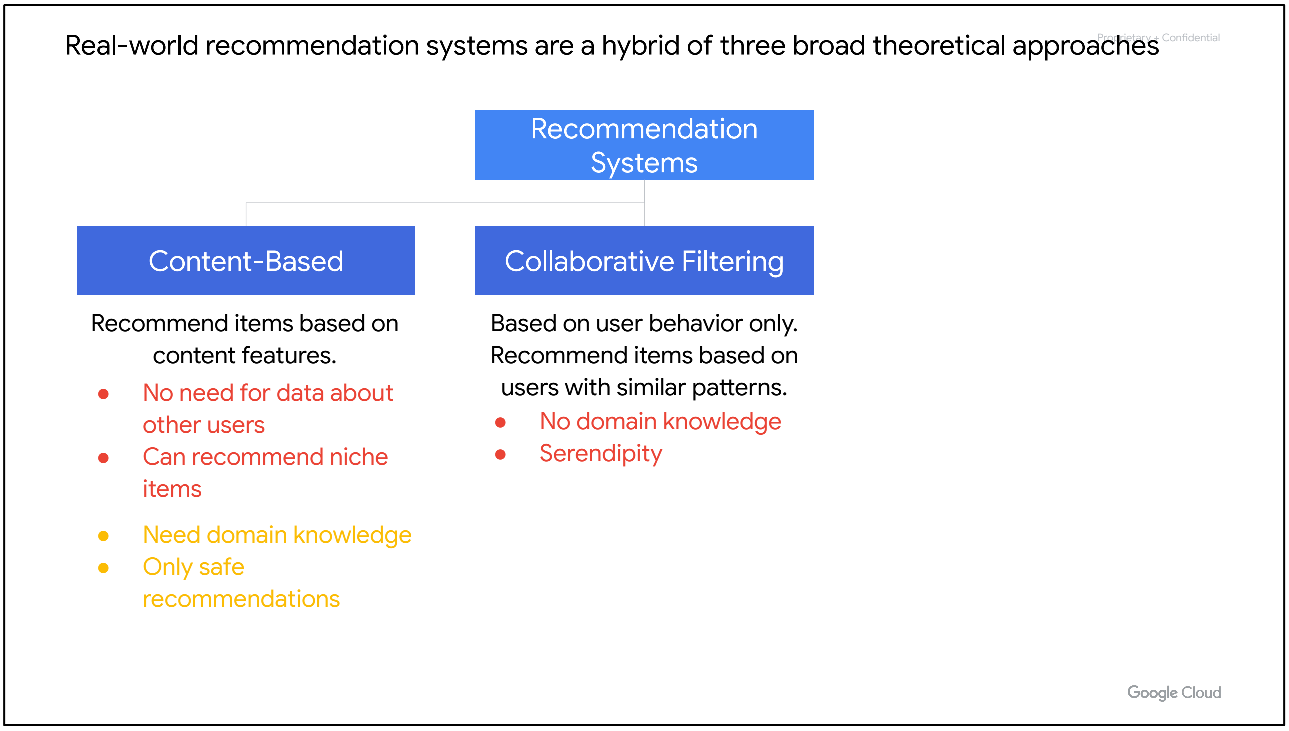
- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge

Collaborative filtering is very powerful because it requires no domain knowledge. The data generated itself simply by users interacting with items, and we can harness that information to predict other favorable user-item interactions through recommending and targeting. Remember, this can be either explicit feedback, such as the number of stars, thumbs up, or a like/dislike button, or it could be implicit feedback such as page views, duration watched, etc. Some systems will have multiple layers of user-interaction that it can take advantage of, because in addition to the classical rating we normally think of, there can be other interactions. An example could be user comments that can be data-mined for sentiment analysis. Often, different types of interaction data can fill the gaps between each other to make a much better overall system.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering

Recommend items based on content features.

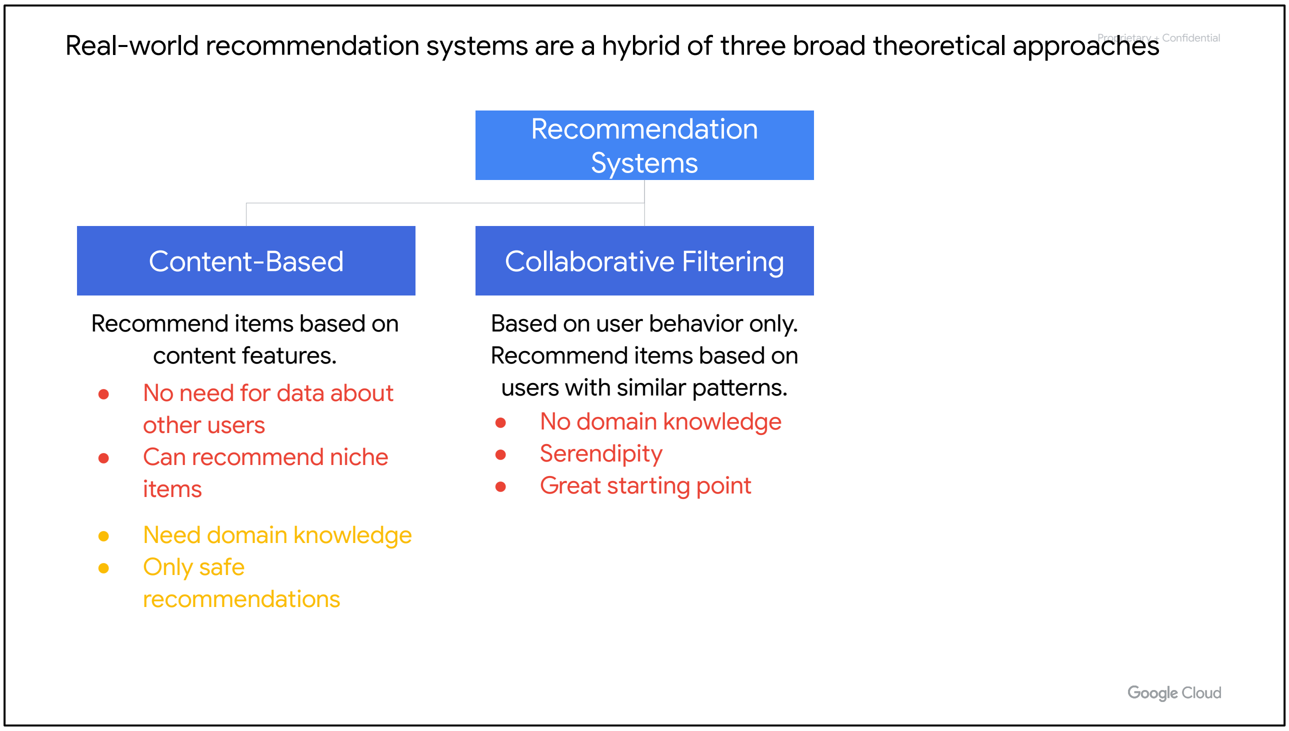
- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity

Collaborative filtering can also solve the problem of of only safe recommendations that is inherent in content-based recommendation systems. This is because not only can collaborative filtering see the user of interest's points in embedding space, it can also tune in to other users' points in embedding space and find similarities between them. For instance, user A might love sci-fi, but has never even thought about seeing anything outside their genre bubble. With collaborative filtering, user A is found to be very similar to user B and user C due to their shared passion for sci-fi. However, user B and C also both love fantasy and action movies, so even though those may be far outside of user A's bubble in embedding space, those might be good recommendations for them to check out.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

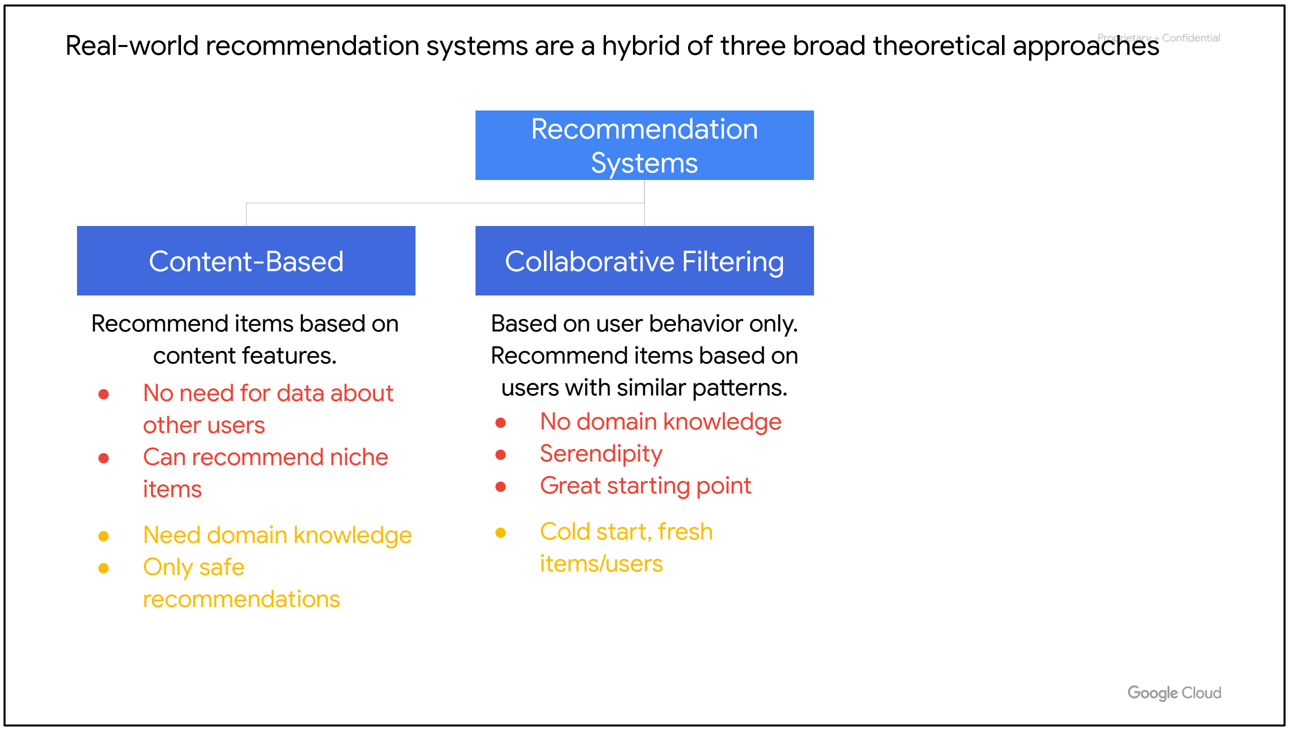
- Need domain knowledge
- Only safe recommendations

Collaborative Filtering

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

Collaborative filtering is also a great starting point. With just a little user-item interaction data, we can create a quick baseline model that we can then check against other models. It can help us find gaps to fill by using other recommendation systems, such as content-based, to make up for the lack of data. Just like the rest of machine learning, it is important to experiment and find what works best.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Collaborative Filtering

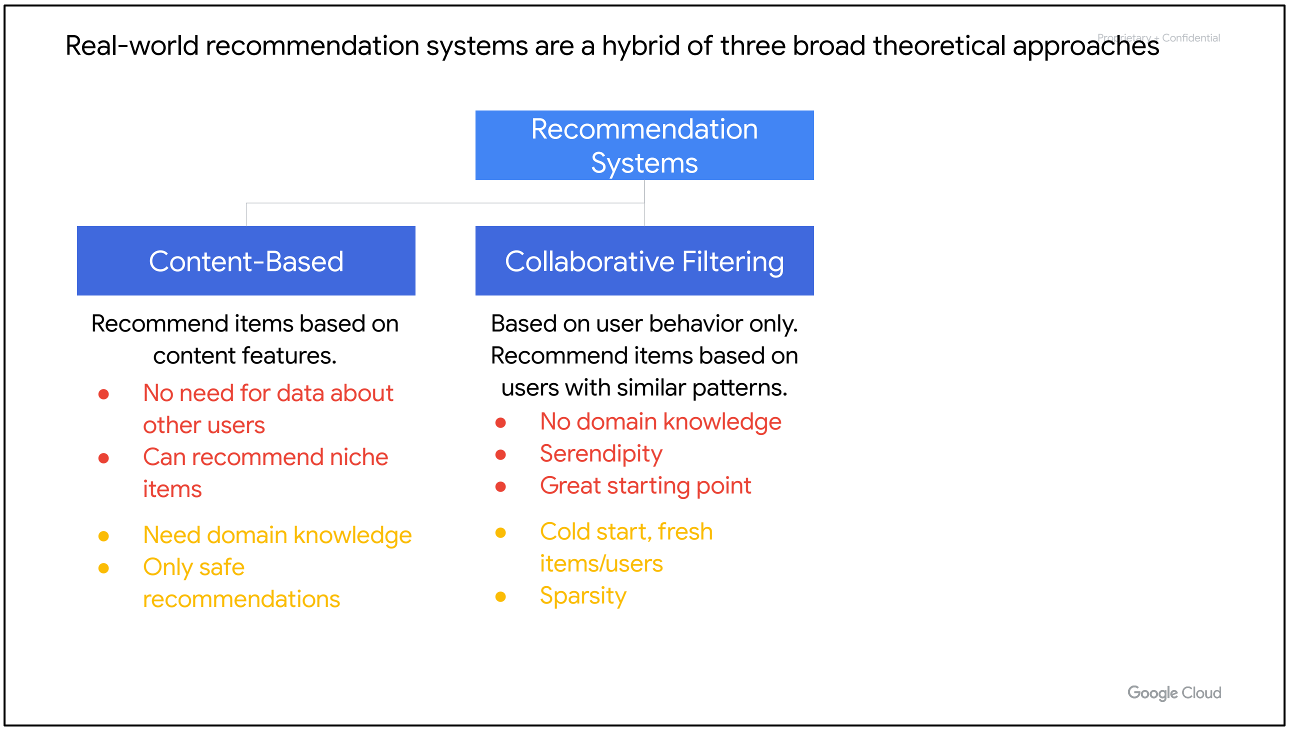
Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

- Cold start, fresh items/users

Just like most things, collaborative filtering isn't a perfect method and does have drawbacks. It mainly suffers from the cold start problem. This happens when we have fresh items or fresh users. For instance, when an item has interacted with a lot of users, we will have a very good idea of what type of users will like that item. When there is little to no interaction data for that item, we don't really have a great idea, because the user sample size is so small or non-existent. We could hopefully use the item embeddings to look nearby and see if those users share any similarities, but a lack of interaction data could be there too.

Same goes for a new user. If they haven't interacted with a lot of items, it is hard to make accurate recommendations. We can use averages of other users or items or even the global average if there is very little overall interaction data. Better yet, we can tap into the other recommendation system types, like content-based, to help fill in the gaps.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

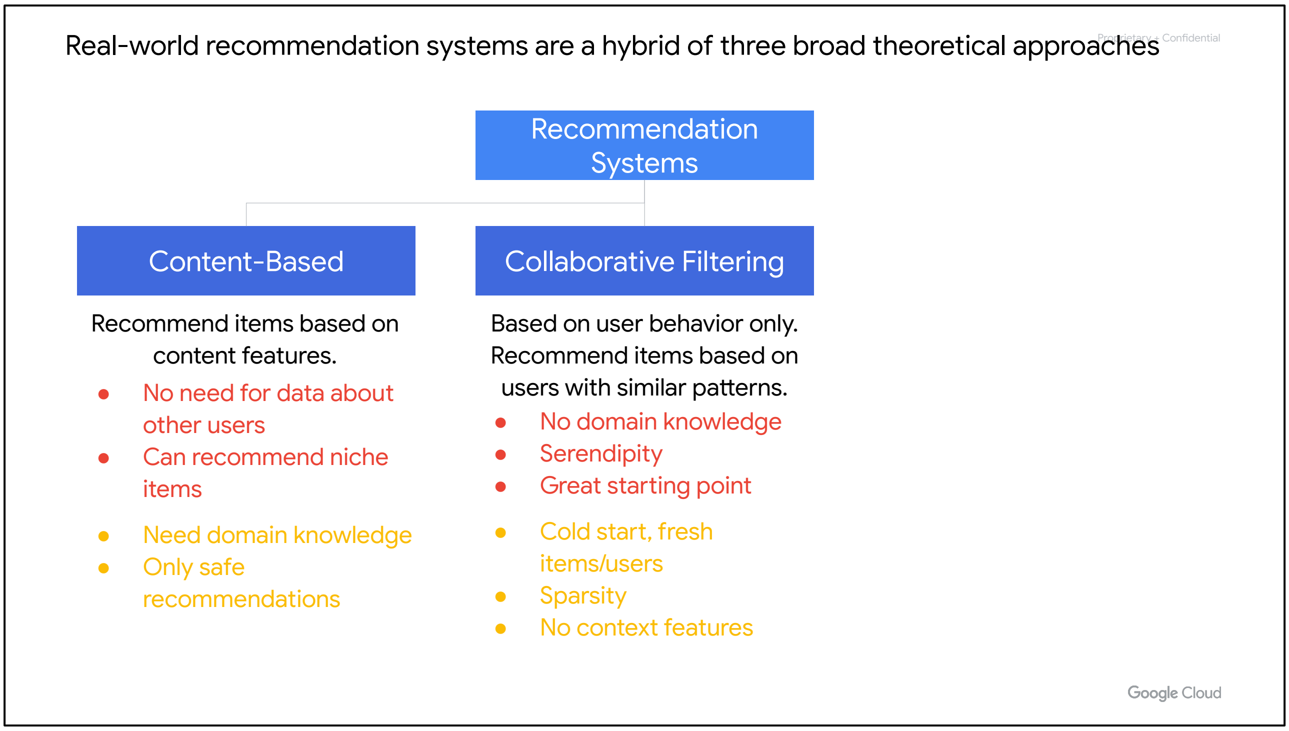
Collaborative Filtering

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

- Cold start, fresh items/users
- Sparsity

This leads right into the problem of sparsity. Remember, matrix factorization collaborative filtering takes our user-item interaction matrix, A, and factorizes it into two hopefully much smaller matrices, U for users, and V for items, each with a dimension of the number of latent factors. It is not as easy to tell when looking at toy problems with very few users and items, but as these both increase, the number of interactions between them become very sparse. Imagine millions of users and thousands or millions of items. Even the most active users will interact with only a small sample of items, and even the most popular items will usually be interacted with by a small subset of users. This can lead to scalability problems later on as well.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

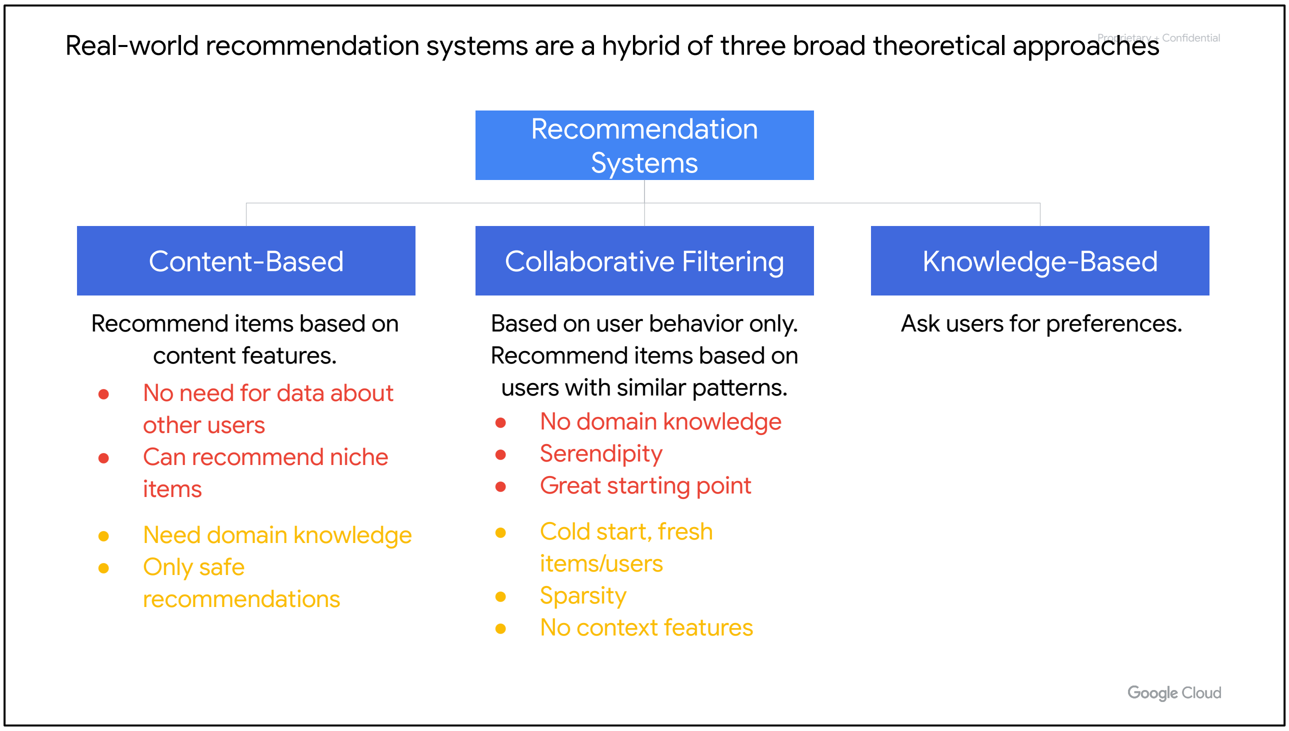
- Need domain knowledge
- Only safe recommendations

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

- Cold start, fresh items/users
- Sparsity
- No context features

Lastly, one of collaborative filtering's pros also leads to one of its cons. It's great that no domain knowledge is needed, but then we have no domain knowledge in our model, which can usually be pretty useful. This lack of context features can reduce the performance of our collaborative filtering models and usually leads us to combining our model with others like content-based recommendation systems.


Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering
Knowledge-Based

Content-Based: Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Collaborative Filtering: Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

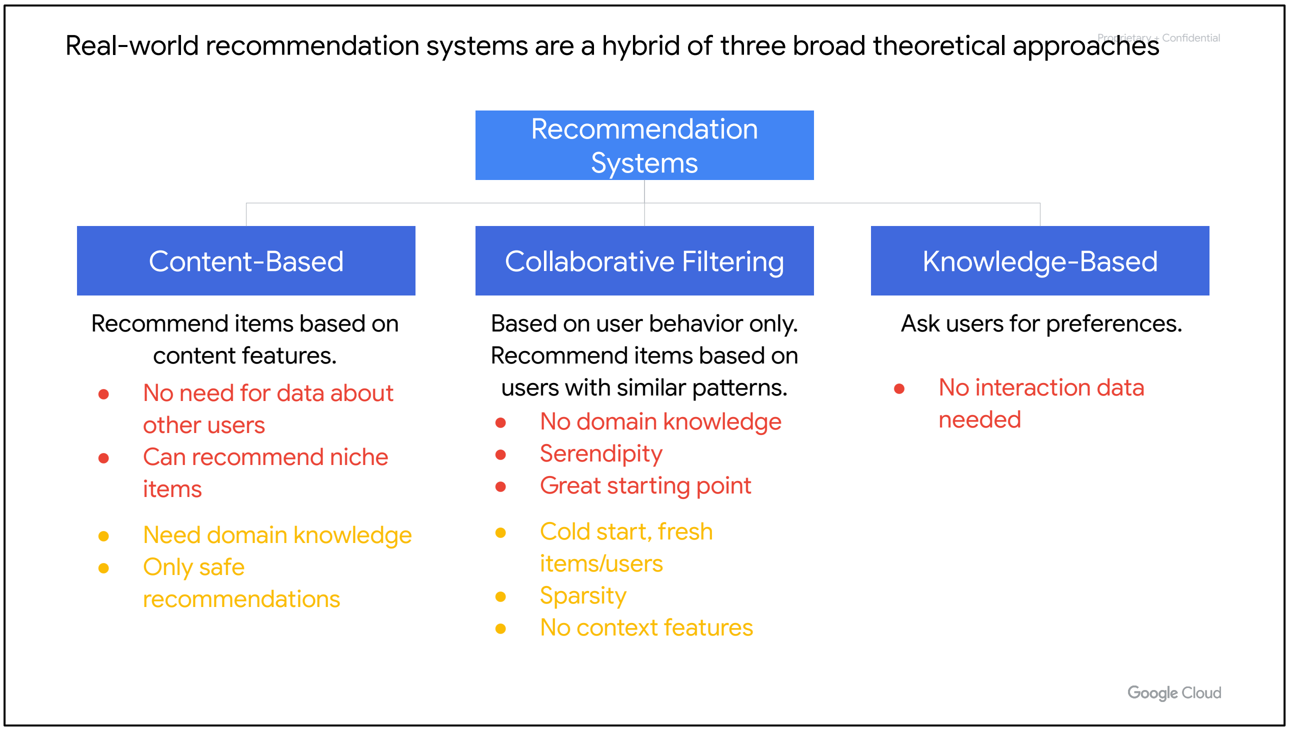
- Cold start, fresh items/users
- Sparsity
- No context features

Knowledge-Based: Ask users for preferences.

We also learned in module 1 about knowledge-based recommendations, where we take either data from user surveys or entered user settings that show user's preferences.

One way of doing this is, assuming it is legal and ethical for your model, to use user-entered data such as where they live, their age, their gender, etc. We use these to try to find similarities. For example, with age, most children will be more likely to prefer what other children prefer, rather than what the elderly enjoy.

We can also ask users. When building knowledge-based recommendation systems, we should keep this in mind when designing them and the point of entry of data. This could be asking users what type of movies they enjoy, what types of food they like, what activities they like doing, etc. This could also be asking users what they don't like so we can filter that out.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering
Knowledge-Based

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Collaborative Filtering

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

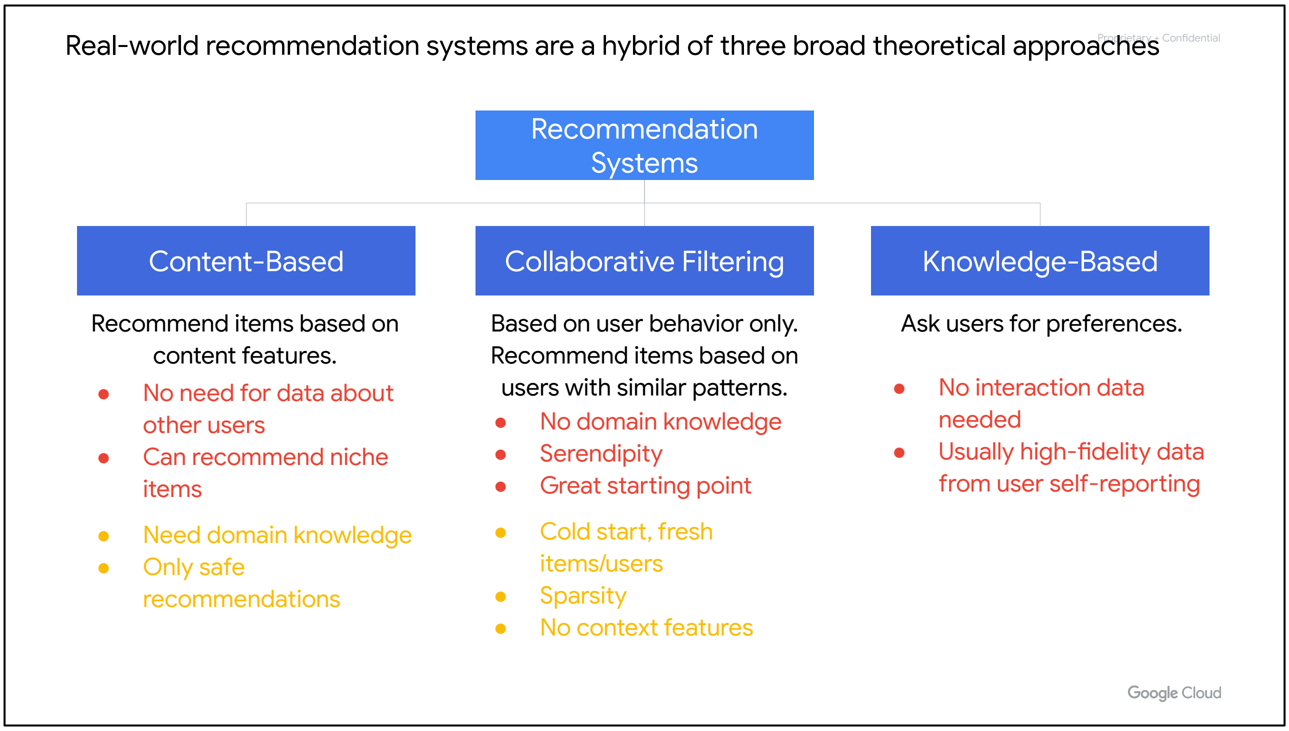
- Cold start, fresh items/users
- Sparsity
- No context features

Knowledge-Based

Ask users for preferences.

- No interaction data needed

A great benefit of knowledge-based recommendation systems is not needing to have user-item interaction data. We simply can rely on user-centric data to link users with other users and recommend similar things that those users liked. This also doesn't require human-generated information about the items, which is usually expensive and hard to generate well without the substantial help of many domain knowledge experts.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems
Content-Based
Collaborative Filtering
Knowledge-Based

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- Need domain knowledge
- Only safe recommendations

Collaborative Filtering

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

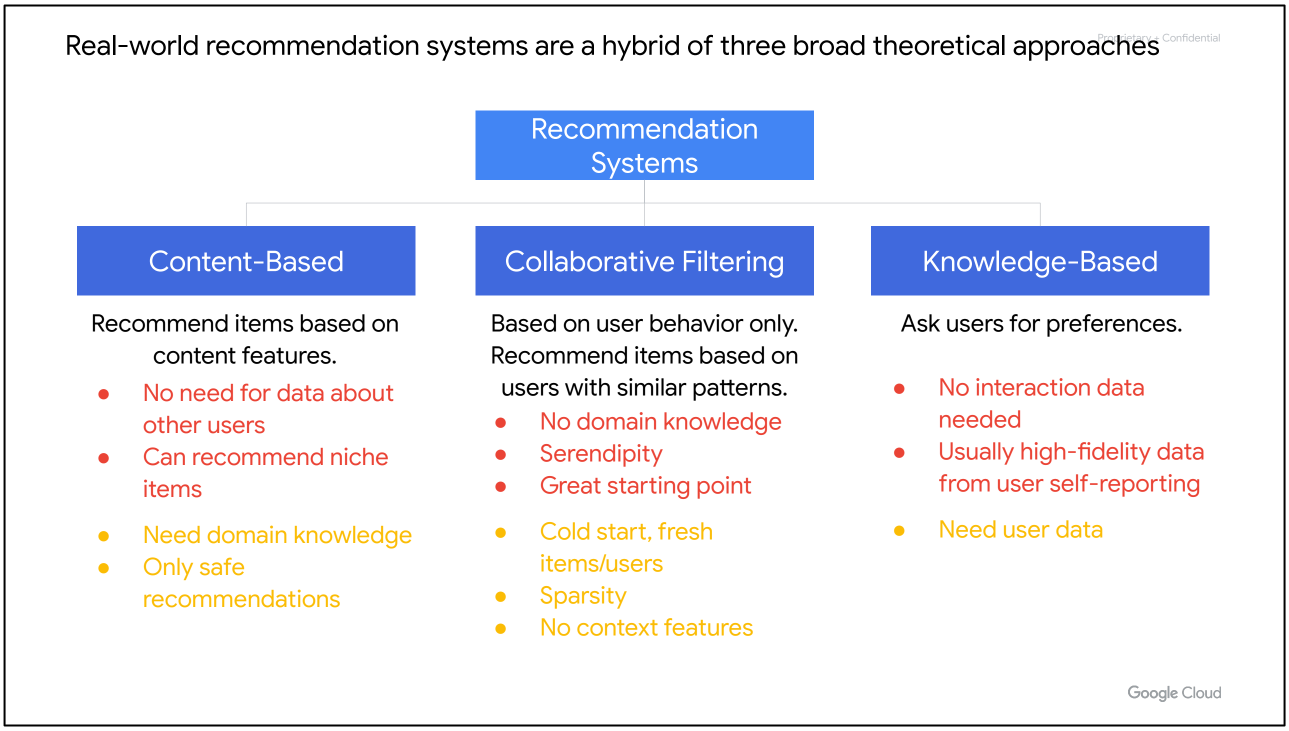
- Cold start, fresh items/users
- Sparsity
- No context features

Knowledge-Based

Ask users for preferences.

- No interaction data needed
- Usually high-fidelity data from user self-reporting

Also, knowledge-based recommendations use data that is of high fidelity, because the user of interest has self-reported their information and preferences, and we can fairly safely assume that those are true. This gives us more trust in the data, because rather than implicitly like some other data, users are explicitly telling us the things they like and don't like.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems

Content-Based

Recommend items based on content features.

- ● No need for data about other users
- ● Can recommend niche items
- ● Need domain knowledge
- ● Only safe recommendations

Collaborative Filtering

Based on user behavior only. Recommend items based on users with similar patterns.

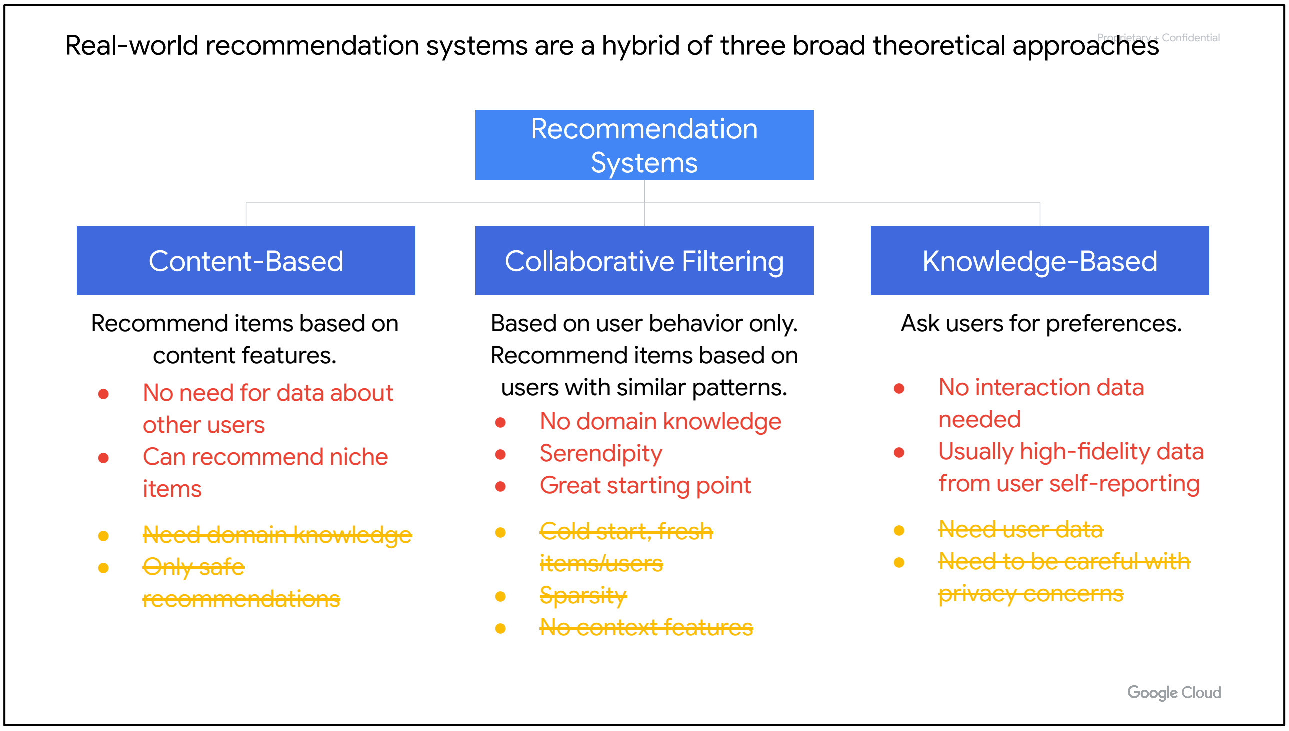
- ● No domain knowledge
- ● Serendipity
- ● Great starting point
- ● Cold start, fresh items/users
- ● Sparsity
- ● No context features

Knowledge-Based

Ask users for preferences.

- ● No interaction data needed
- ● Usually high-fidelity data from user self-reporting
- ● Need user data

Unfortunately, knowledge-based systems don't work well if users don't select their preferences or set their properties. Just like all machine learning, a model is going to really struggle if there is a major lack of data. The lack of user data, however, can motivate how we design our collection processes. Perhaps we were asking users the wrong questions or weren't asking the right ones? Maybe we didn't create enough fields in the profile page for users to fill out their information? Maybe users don't feel comfortable sharing their preferences with us, and we have a messaging problem? As you can see, there can be a myriad of problems, but there are many possible solutions as well.



Real-world recommendation systems are a hybrid of three broad theoretical approaches

Recommendation Systems

Content-Based

Recommend items based on content features.

- No need for data about other users
- Can recommend niche items

- ~~Need domain knowledge~~
- ~~Only safe recommendations~~

Collaborative Filtering

Based on user behavior only. Recommend items based on users with similar patterns.

- No domain knowledge
- Serendipity
- Great starting point

- ~~Cold start, fresh items/users~~
- ~~Sparsity~~
- ~~No context features~~

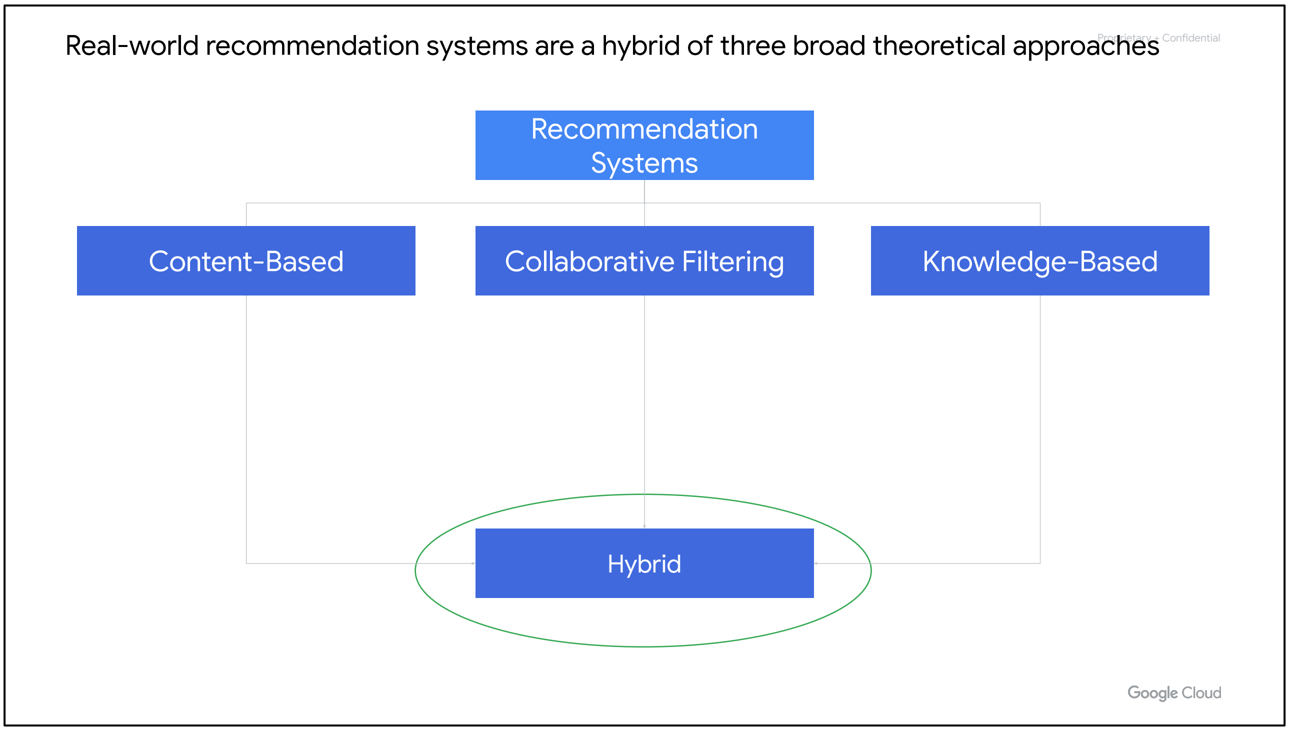
Knowledge-Based

Ask users for preferences.

- No interaction data needed
- Usually high-fidelity data from user self-reporting

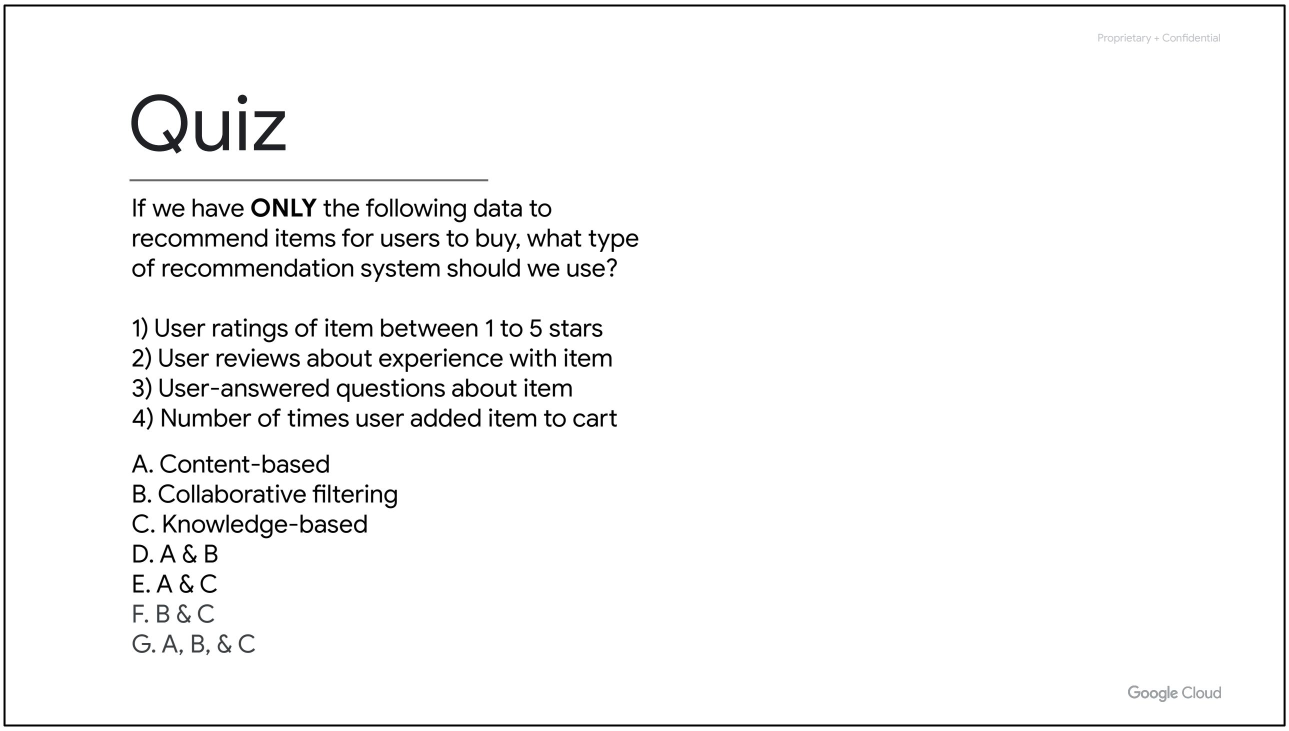
- ~~Need user data~~
- ~~Need to be careful with privacy concerns~~

After going through all of the strengths and weaknesses of these three recommendation system types, the next question is obvious. How can we keep the strengths, and get rid of the weaknesses? Well fortunately, there is a solution for that!



Real-world recommendation systems are a hybrid of three broad theoretical approaches

That solution is using hybrid recommendation systems. These might sound more intimidating than they actually are. They don't have to be super complex and can be rather simple. Imagine training content-based, collaborative filtering, and knowledge-based recommendation systems that each make a recommendation of an item for a user. All three of these models might recommend different items, and some predictions may be better than others due to things like data size, quality, and model properties. A super simple way to create a hybrid model is to just take things from each of the models, and combine them all in a neural network. The idea is that the independent errors within each model will cancel out, and we'll have much better recommendations. We will soon see several examples of this.



Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

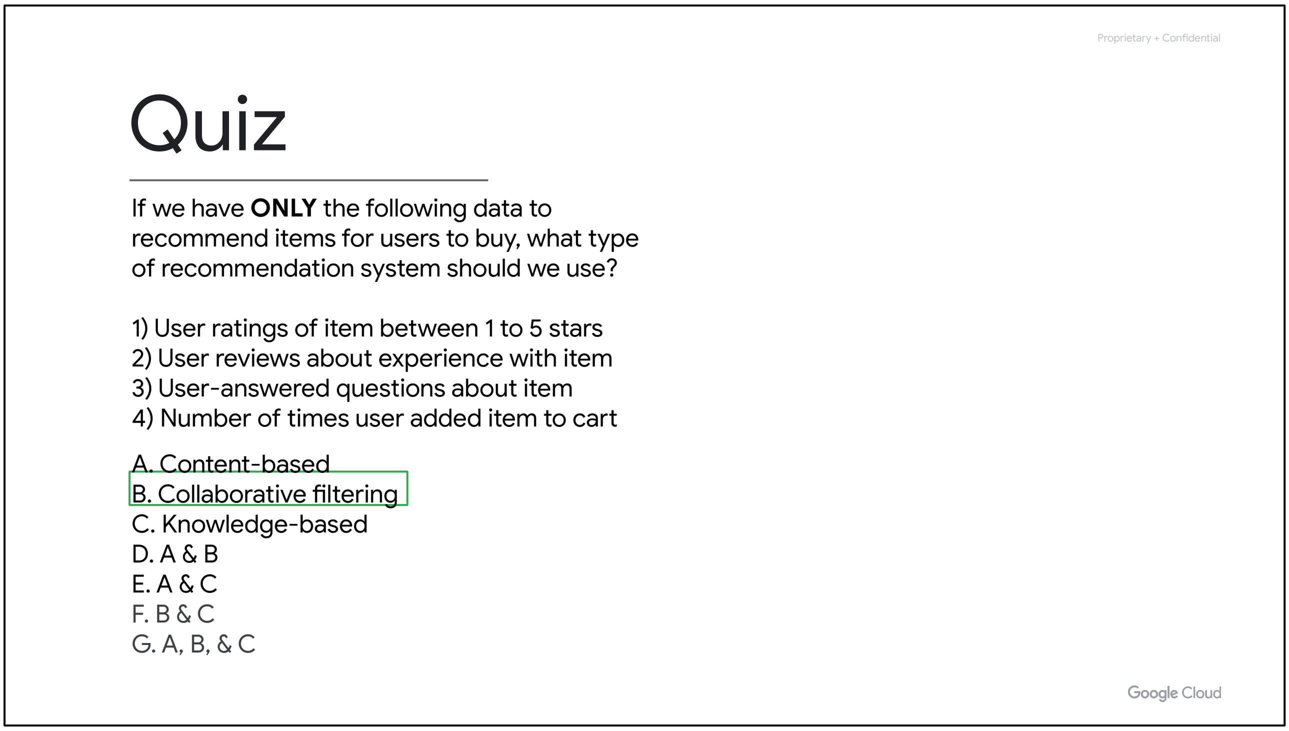
1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Let's now test our knowledge! We've refreshed ourselves about three popular types of recommendation systems, and the pros and cons of each. We've also touched on how hybrid models can use a combination of them to produce even better recommendations than each separate model could on its own.

If we have ONLY the following data to recommend items for users to buy, what type of recommendation system should we use?

User ratings of item between 1 to 5 stars, user comments about experience with item, user-answered questions about item, and the number of times user added item to cart.



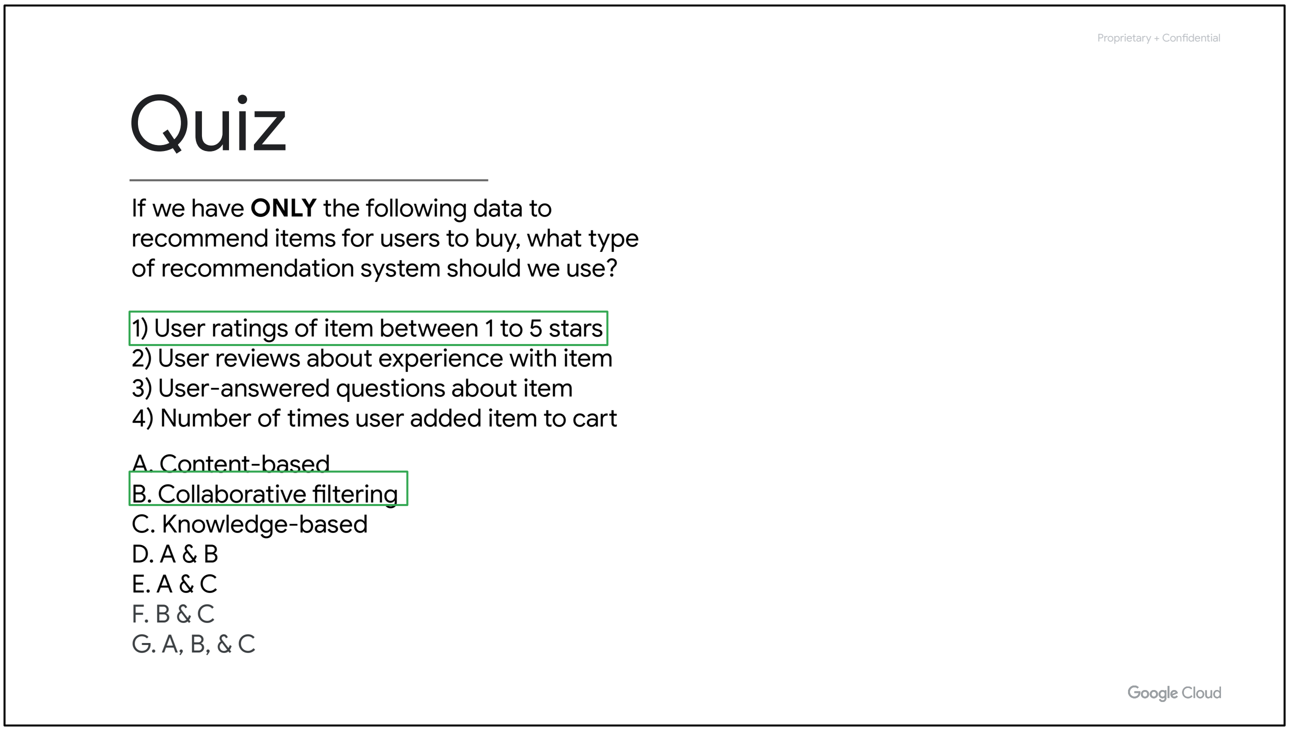
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
[✓ B. Collaborative filtering]
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

The correct answer is B! Most people probably jumped straight to answer G thinking that a hybrid model is always the answer, and that is usually true, but that might not be possible in this hypothetical example. Let's go over the reasoning behind this.



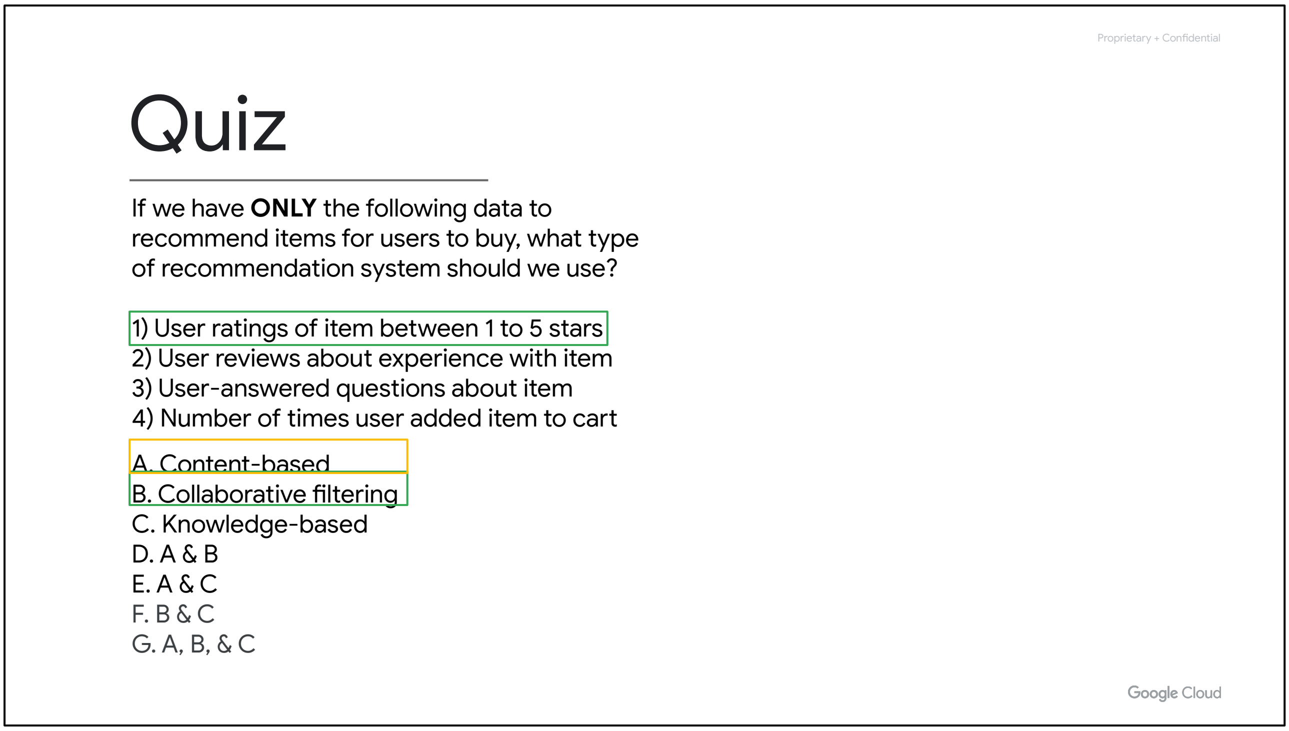
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

[✓ 1) User ratings of item between 1 to 5 stars]
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
[✓ B. Collaborative filtering]
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Let's look at our first dataset. We have user ratings of items that are scores between 1 to 5 stars. Well, this is explicit feedback of user-item interactions, so the very first thing that comes to mind is collaborative filtering, which can use a matrix factorization algorithm like weighted alternating least squares, or WALS, to train.



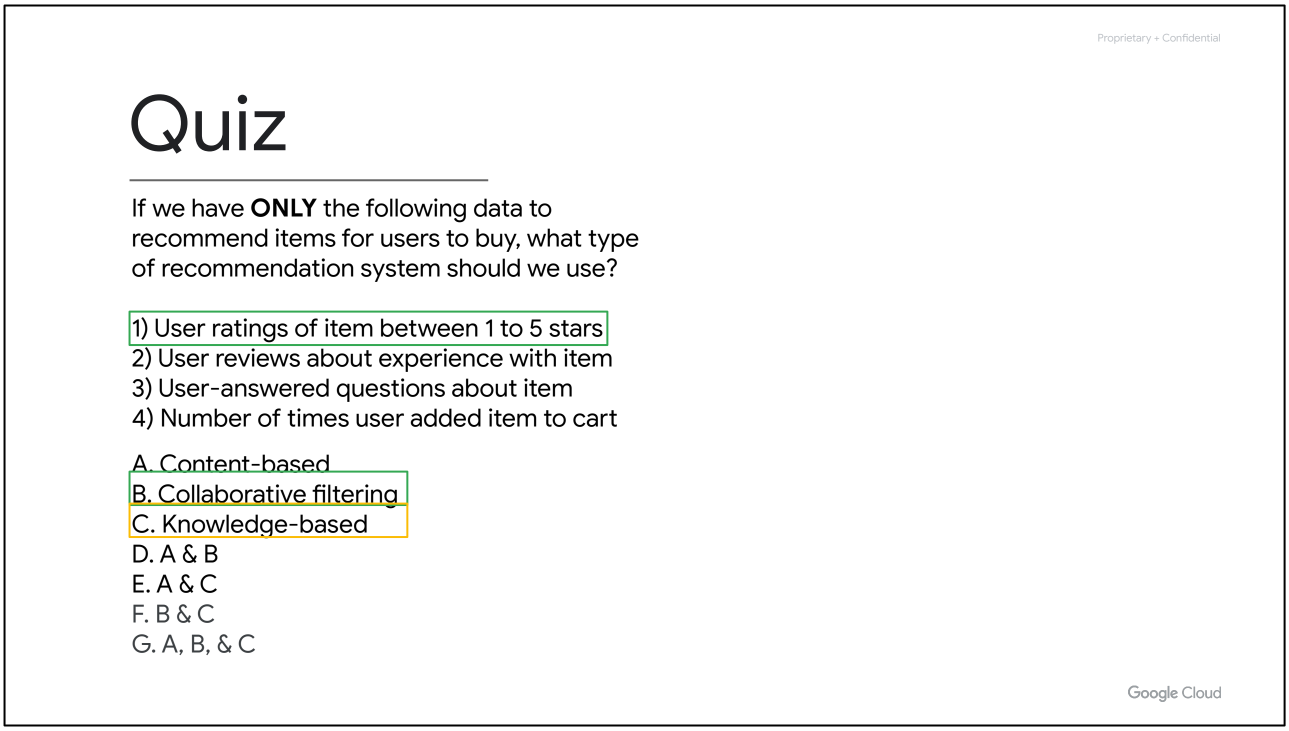
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

This isn't content-based because this dataset is not metadata about the item.



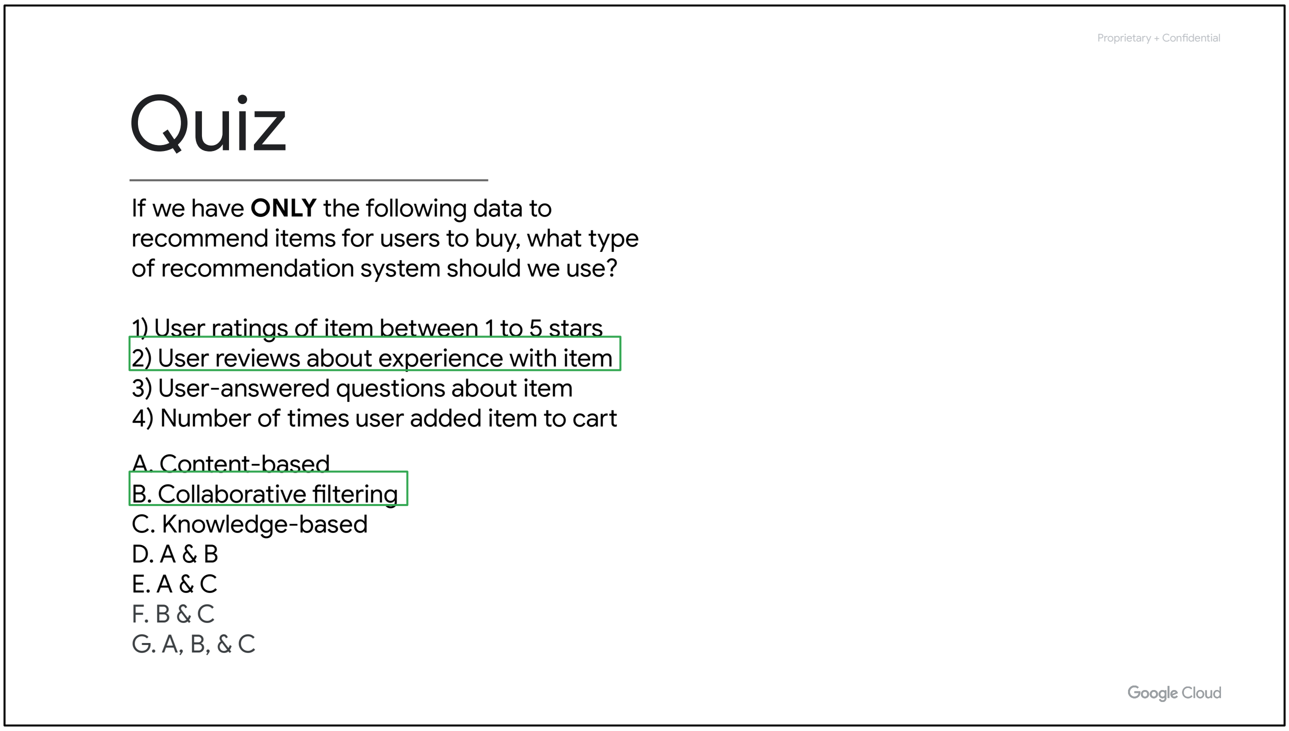
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

And it's not knowledge-based, because it isn't any personal user or user preference data.


Proprietary + Confidential

Quiz

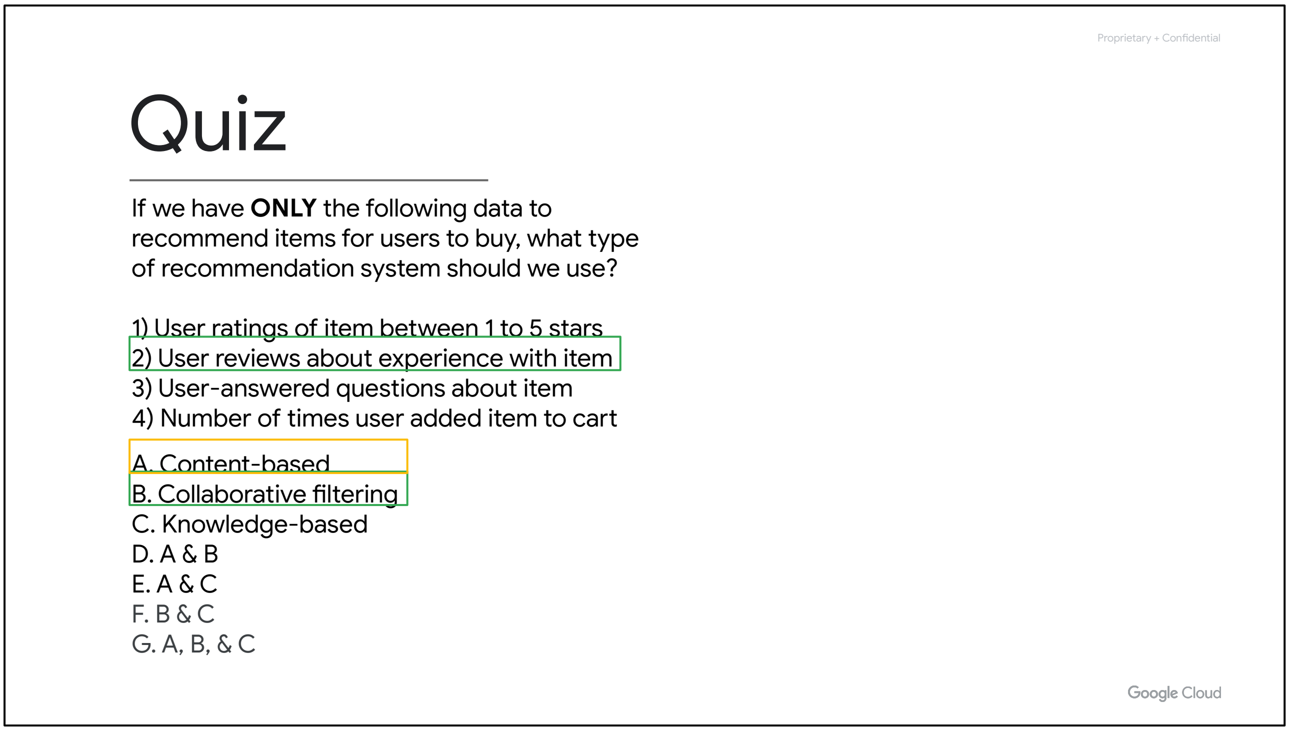
If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Google Cloud

Now on to the second dataset, user reviews about experience with item. Well, we might first think, ah yes, it is text, so we could embed the text and use a content-based recommendation system on that. And because, based on the first dataset needing collaborative filtering, then the answer is of course, use a hybrid model! We need to stop and think more closely.


Proprietary + Confidential

Quiz

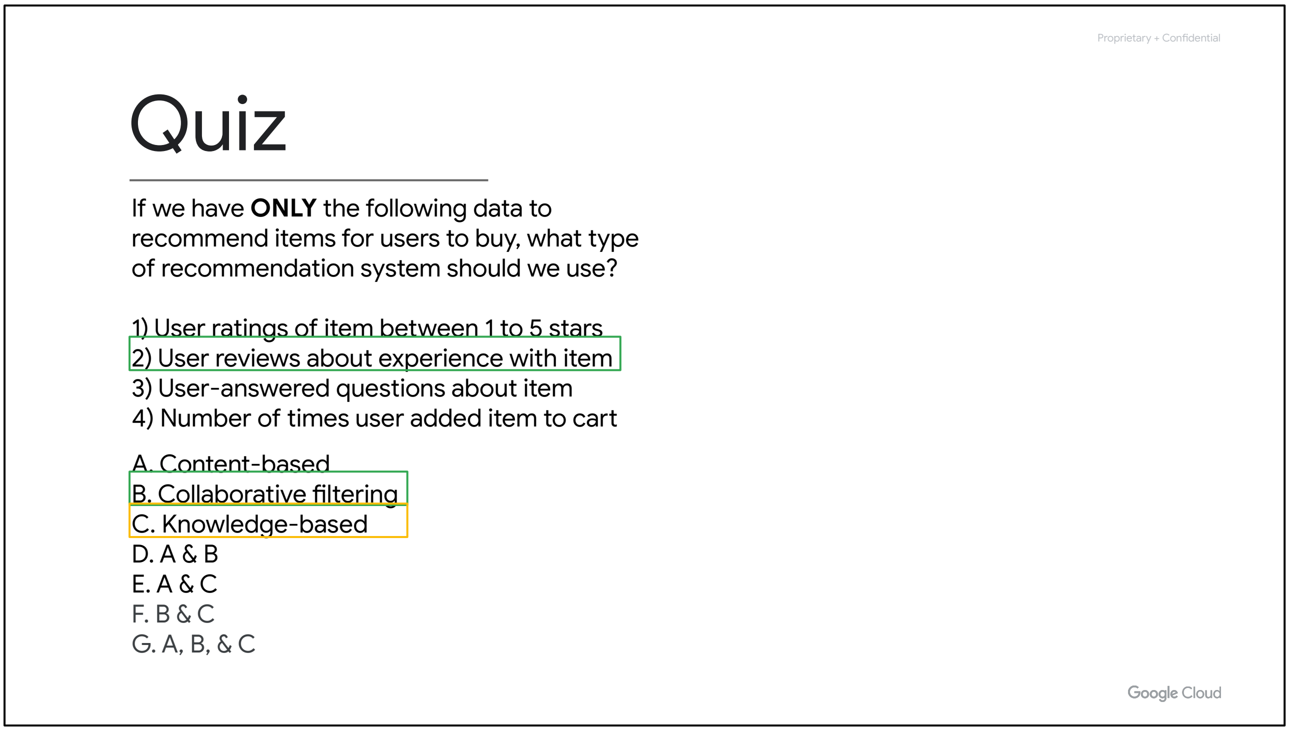
If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Google Cloud

This isn't a description or something written by a product expert, but from a user who has interacted with it. It's starting to sound less like content-based and more like collaborative filtering, but this time using unstructured data. We can indeed put this into an embedding and use that or perform sentiment analysis on it, and then create a user-item interaction matrix out of that.



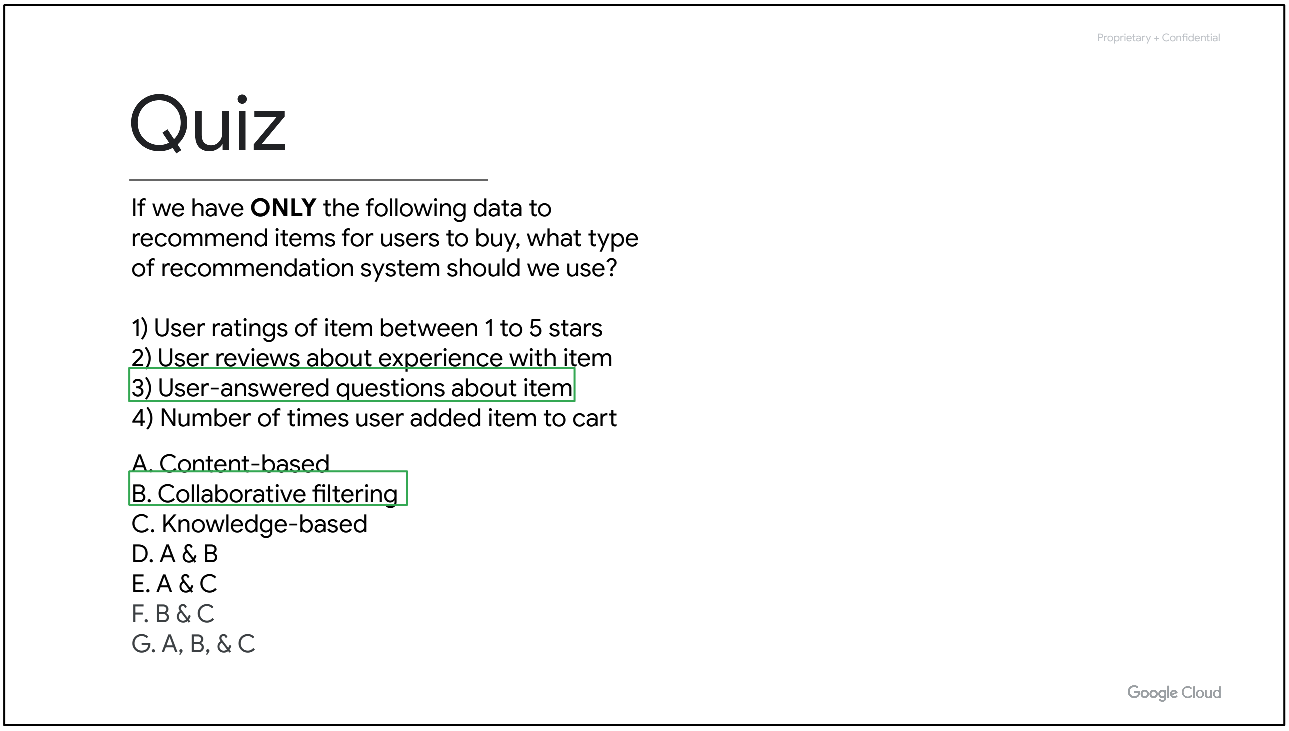
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

It also is still not knowledge-based, because the data doesn't contain any user-centric or global user preference information.


Proprietary + Confidential

Quiz

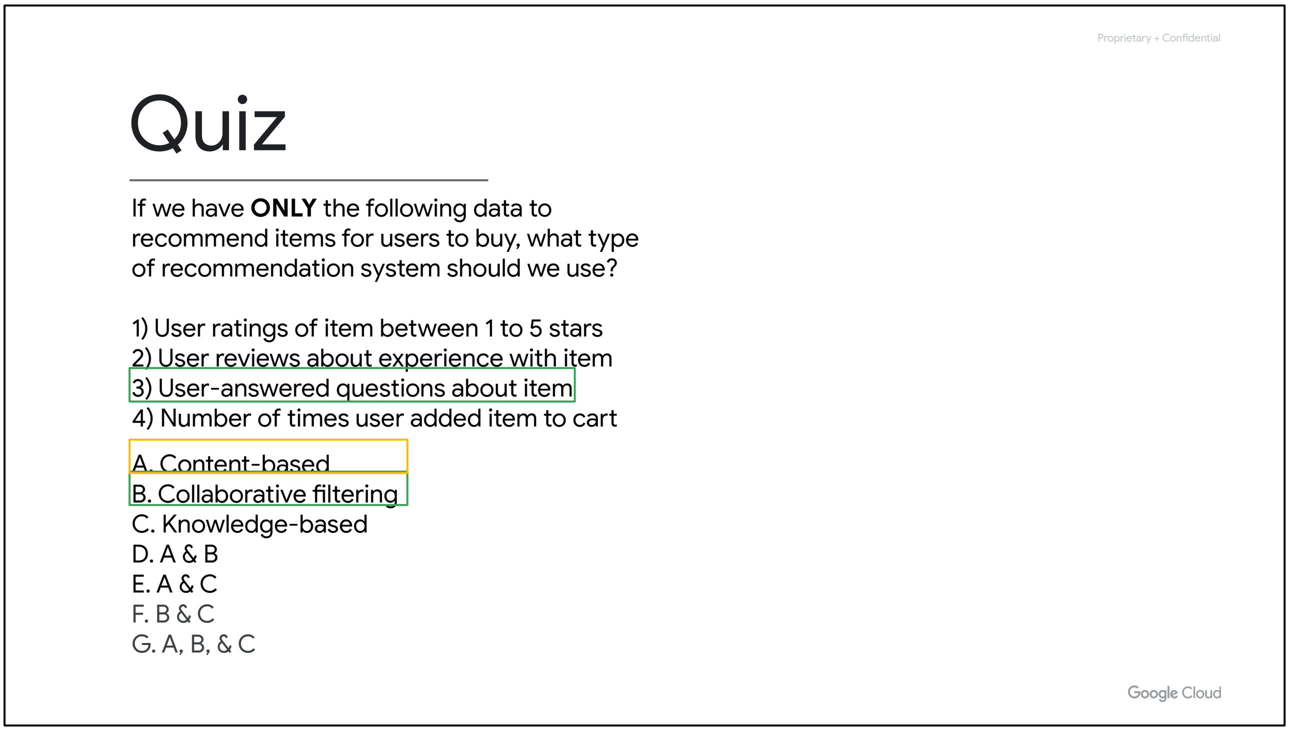
If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Google Cloud

So the third dataset has user-answered questions about items that other users can read when judging whether to purchase an item. It looks very similar to the previous dataset by being free text; however, this feedback is quite a bit less explicit. If we were to perform sentiment analysis, most of the answers might be neutral because users are just being factual about the item. Of course, some users might slip in some praise or dislike into their answers, but it might not be enough to go on. We can, however, use this as implicit feedback, because we would assume that, if a person is responding about an item, they have one or more of their own and like it enough to answer questions for others. Remember, implicit feedback usually involves an assumption, which could turn out to be wrong. Perhaps users dislike the item so much that they go out of their way to warn people in their answers to questions.


Proprietary + Confidential

Quiz

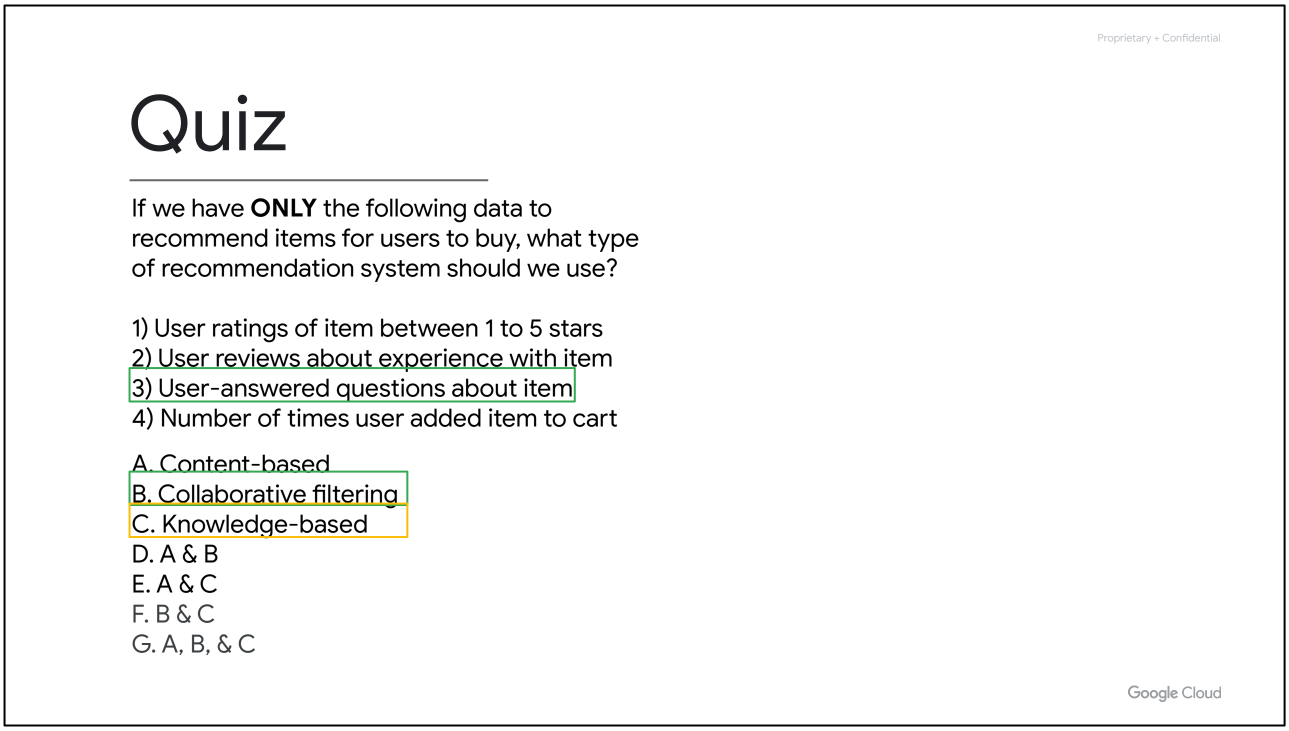
If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Google Cloud

This still probably isn't content-based, because it is the user clarifying the product details to the best of their ability, and not a product expert who definitely should know the ground truth.



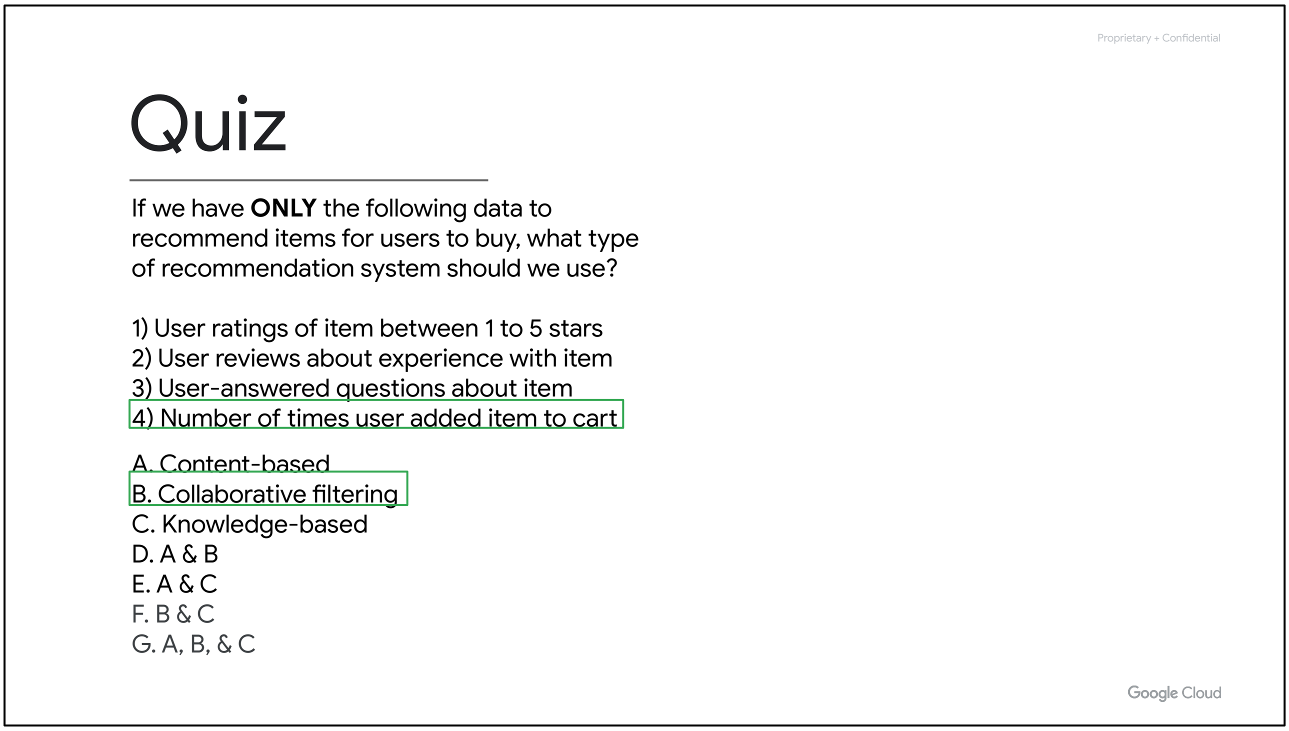
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Once again this is not knowledge-based because there is no user-centric data



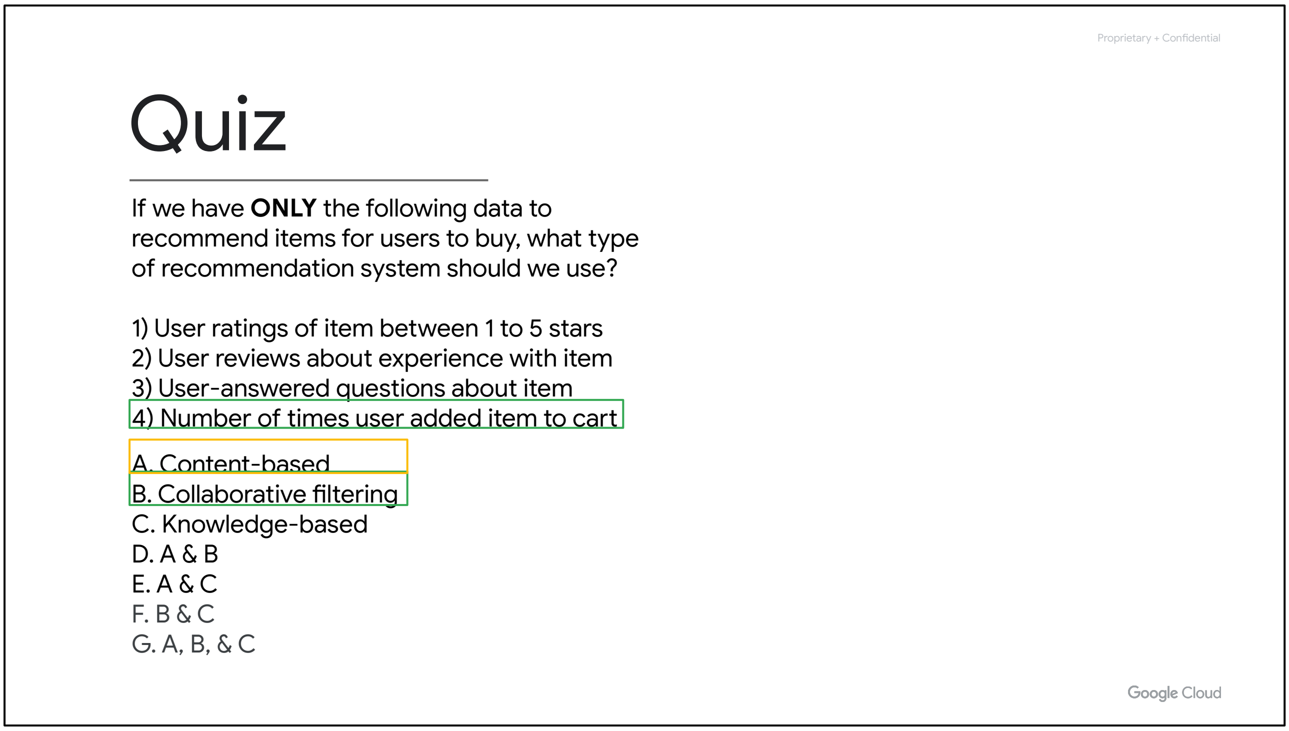
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
[✓ B. Collaborative filtering]
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Last but not least, we have a dataset of the number of times users added items to their online shopping cart. Once again, this is user-item interaction data, so we'll use collaborative filtering. However, let's analyze the type of feedback it is. So the first thought that jumps out at us is that it is implicit feedback, because this is not an explicit rating. The user is not explicitly telling us how much they liked or disliked the item, therefore it must be implicit feedback. This is true, but it could also be explicit feedback if this dataset includes whether users checked out. We could make the assumption that, if users add an item to their cart AND then purchase it, not just once, but multiple times, they must like or at least need the item. There are probably not many users continually buying items they don't want or need.



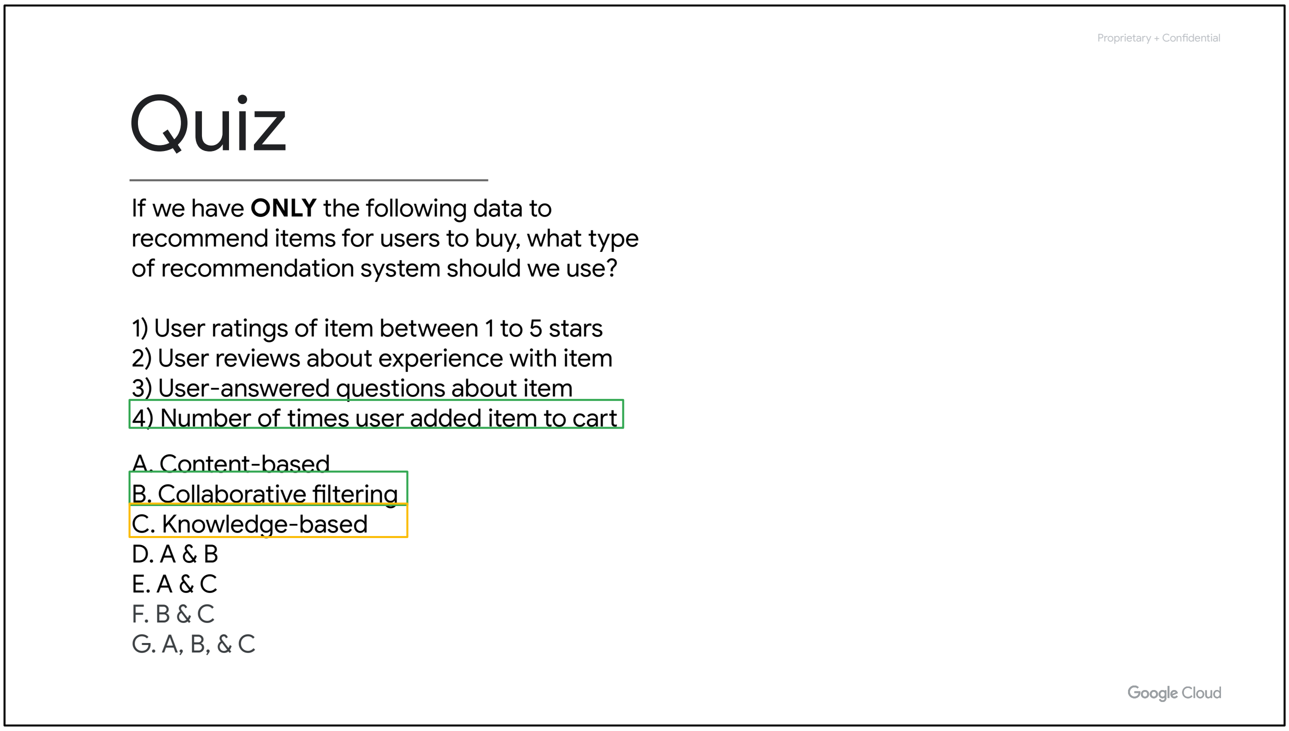
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Again, this isn't content-based because it is not metadata about the item.


Proprietary + Confidential

Quiz

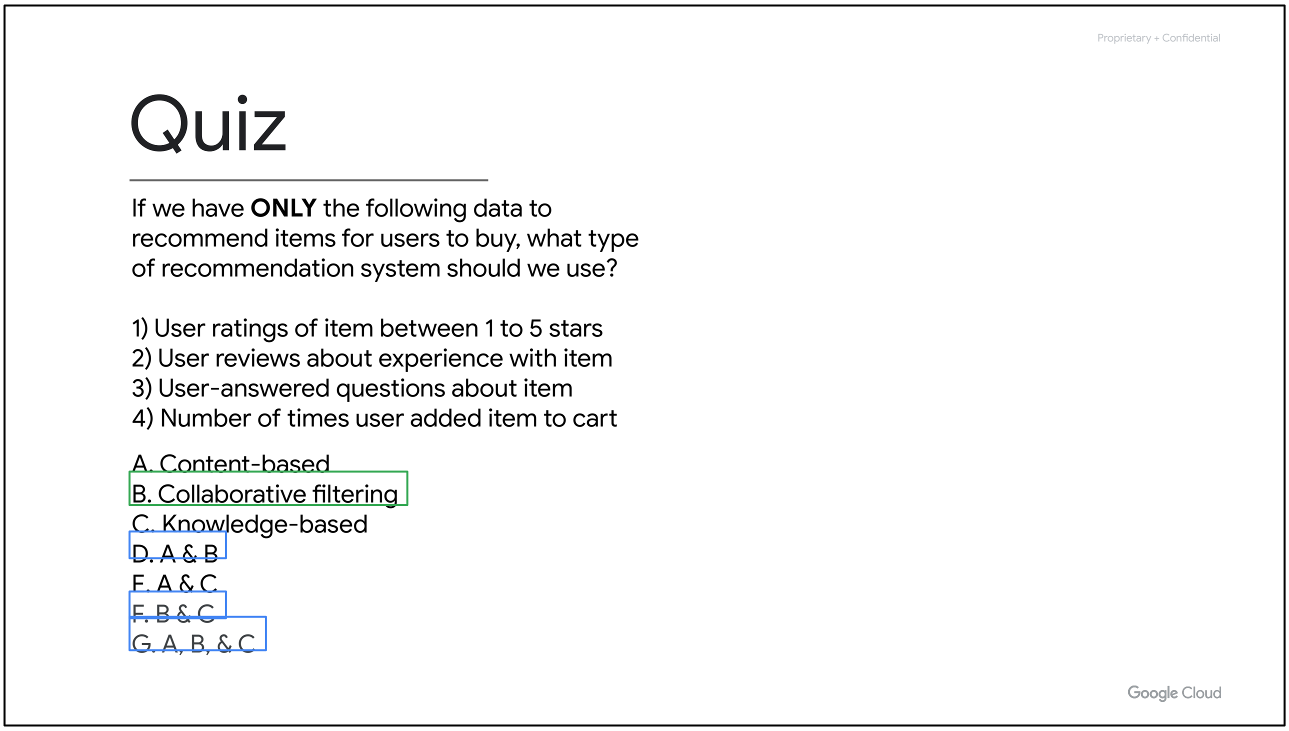
If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
B. Collaborative filtering
C. Knowledge-based
D. A & B
E. A & C
F. B & C
G. A, B, & C

Google Cloud

There's still no user only information in this dataset, so that's a no for knowledge-based.



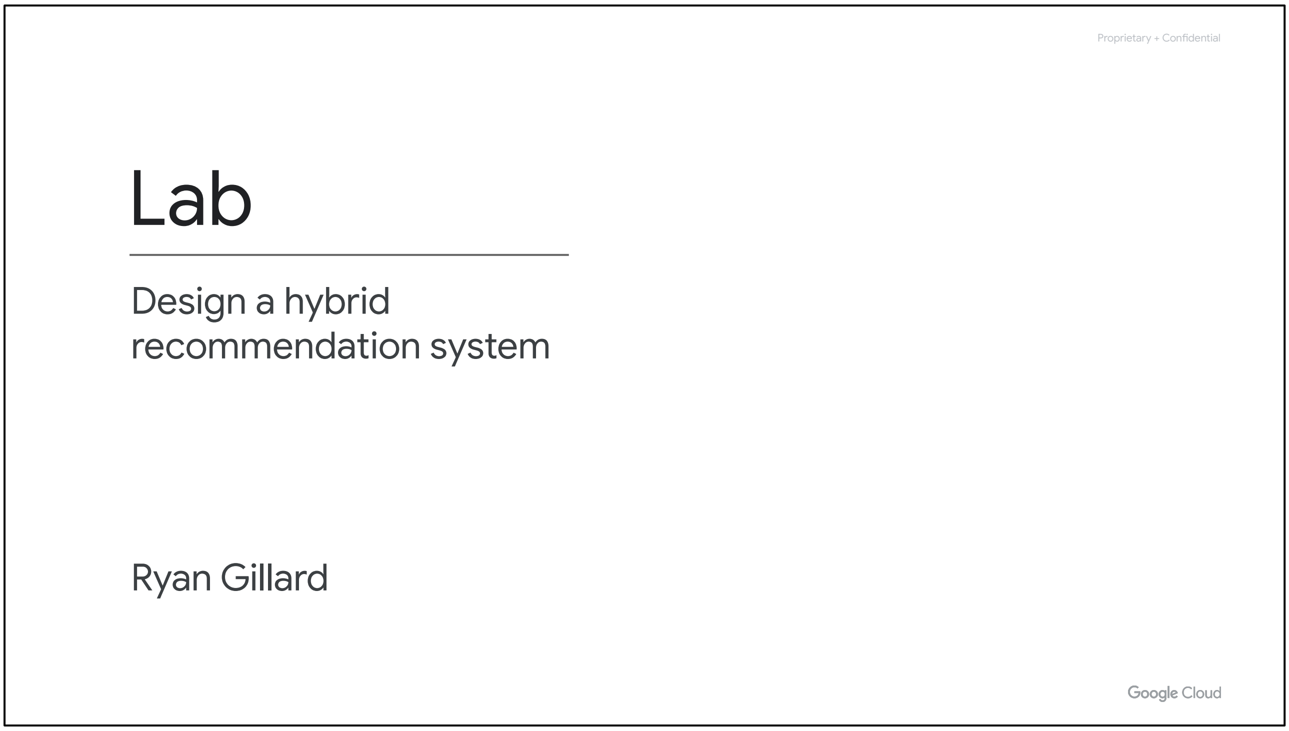
Quiz

If we have **ONLY** the following data to recommend items for users to buy, what type of recommendation system should we use?

1) User ratings of item between 1 to 5 stars
2) User reviews about experience with item
3) User-answered questions about item
4) Number of times user added item to cart

A. Content-based
[✓ B. Collaborative filtering]
C. Knowledge-based
[✓ D. A & B]
E. A & C
[✓ F. B & C]
[✓ G. A, B, & C]

Of course, if we had data about the items themselves and data about our users and/or their self-reported preferences, a hybrid approach would probably give the best results. We still wouldn't choose answer E, because it excludes collaborative filtering, which—assuming our 4 given datasets are good—we definitely will include in our hybrid model.

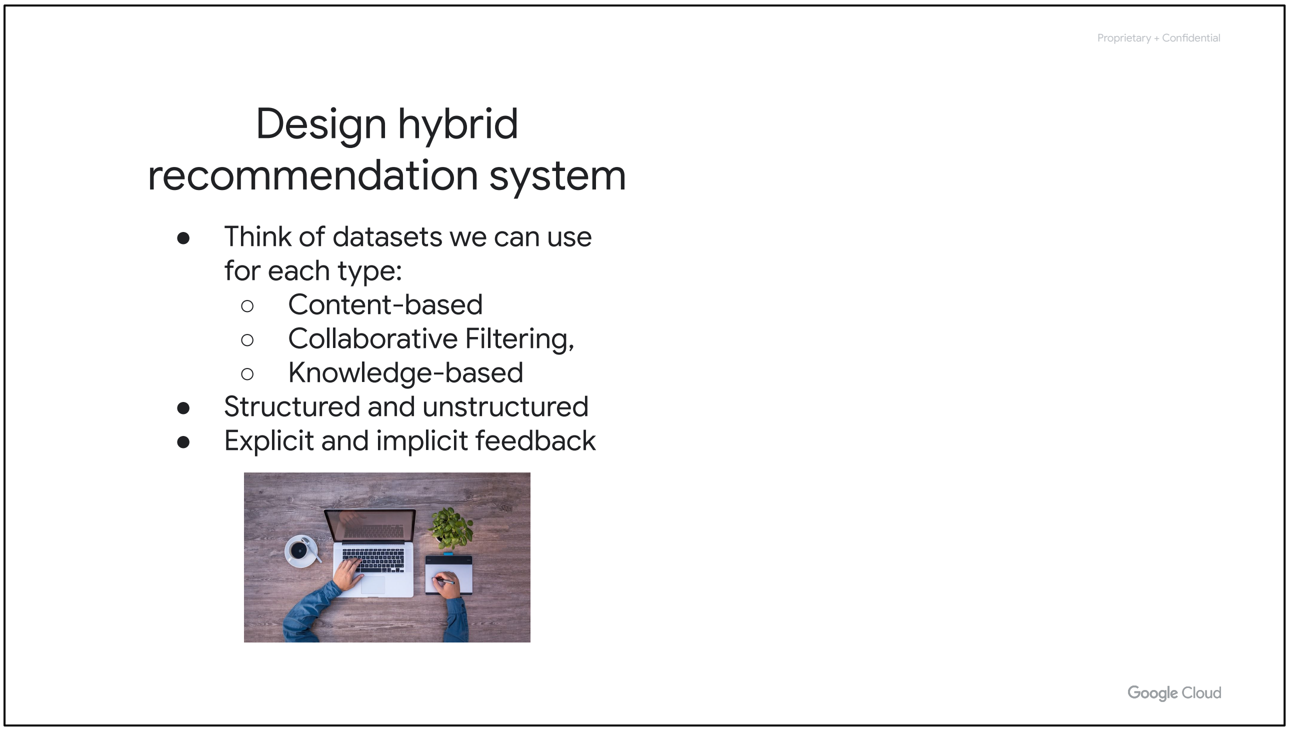


Lab

Design a hybrid recommendation system

Ryan Gillard

Now it's our turn! In this lab, we are going to design a hybrid recommendation system to recommend movies to users. We'll come up with some useful datasets that we think are important to collect. There are many ways to connect things together, so experimentation will help find one of the better combinations.



Design hybrid recommendation system

- Think of datasets we can use for each type:
  - Content-based
  - Collaborative Filtering,
  - Knowledge-based
- Structured and unstructured
- Explicit and implicit feedback

Google Cloud

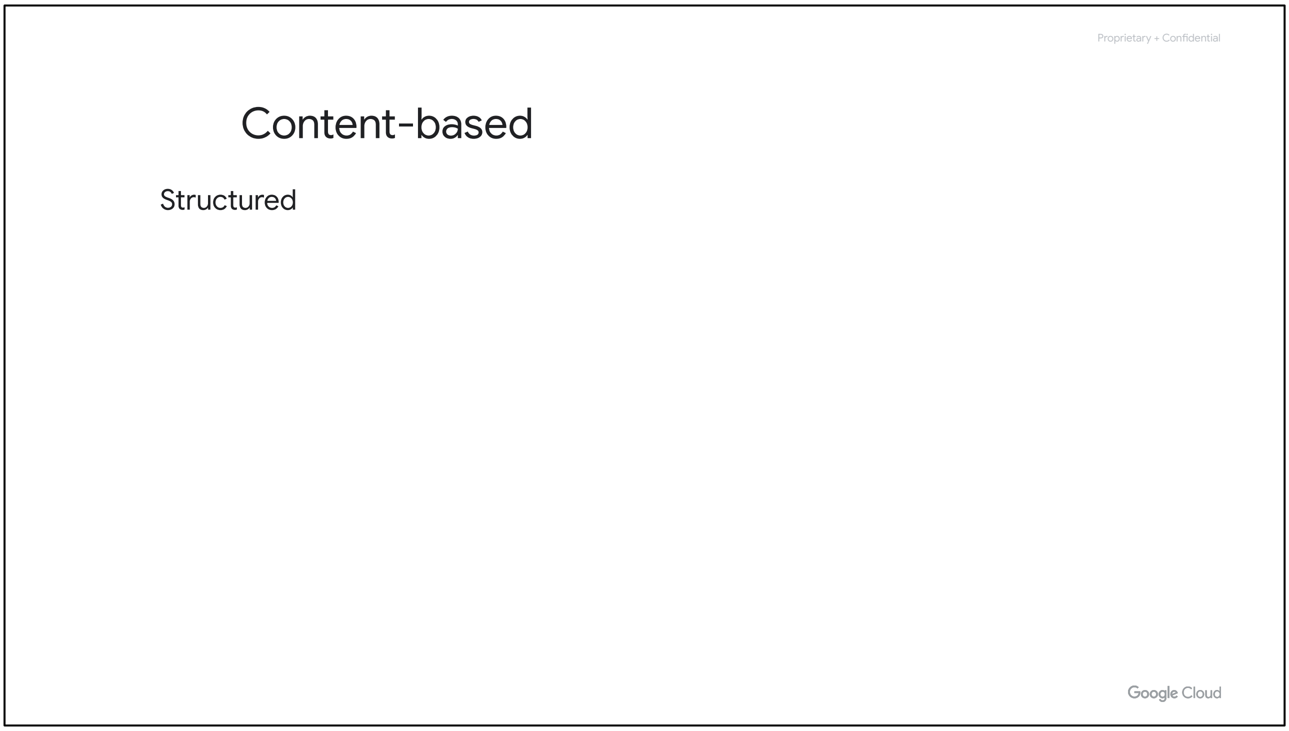
In this lab, we want to have a fun thought experiment on how we can best recommend movies to users by using a hybrid recommendation system. Think of several datasets that would be useful for each recommendation system type: content-based, collaborative filtering, and knowledge-based. Remember, there are both structured and unstructured data sources we can use from our website as well as both explicit and implicit types of feedback.

https://pixabay.com/en/laptop-mockup-graphics-tablet-2838921/ cc0

First, we're going to look at what datasets would be important to best recommend movies to users, for content-based recommendation models. Think about what type of data would be available on a movie's page on our website. Remember, it can be both structured and unstructured. Also, this is content-based, so it shouldn't really involve the users, but should be from experts with domain knowledge about the item.

So now take a moment and think, write down, or discuss with your colleagues what datasets we could use for content-based recommendations and then we'll go through some of the ones we came up with.

[PAUSE]



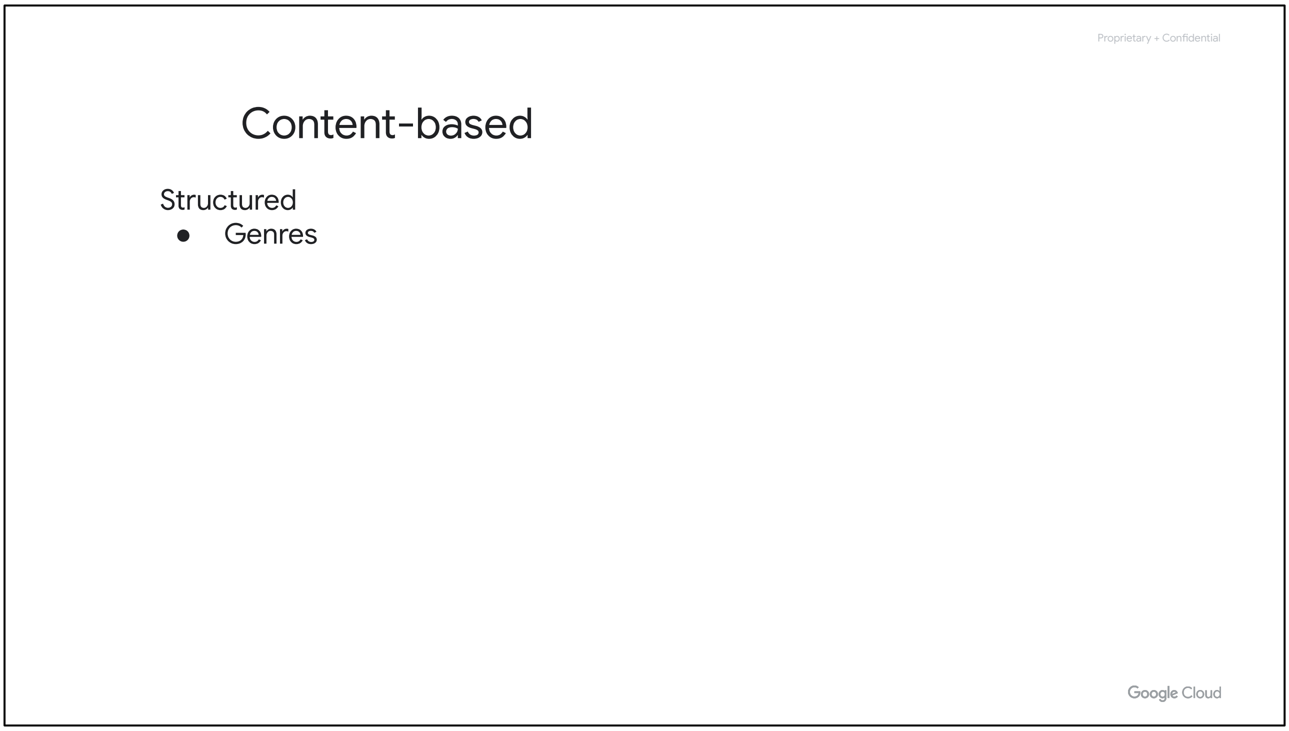
Proprietary + Confidential

Content-based

Structured

Google Cloud

All right, let's see some of the ideas we came up with.



Proprietary + Confidential

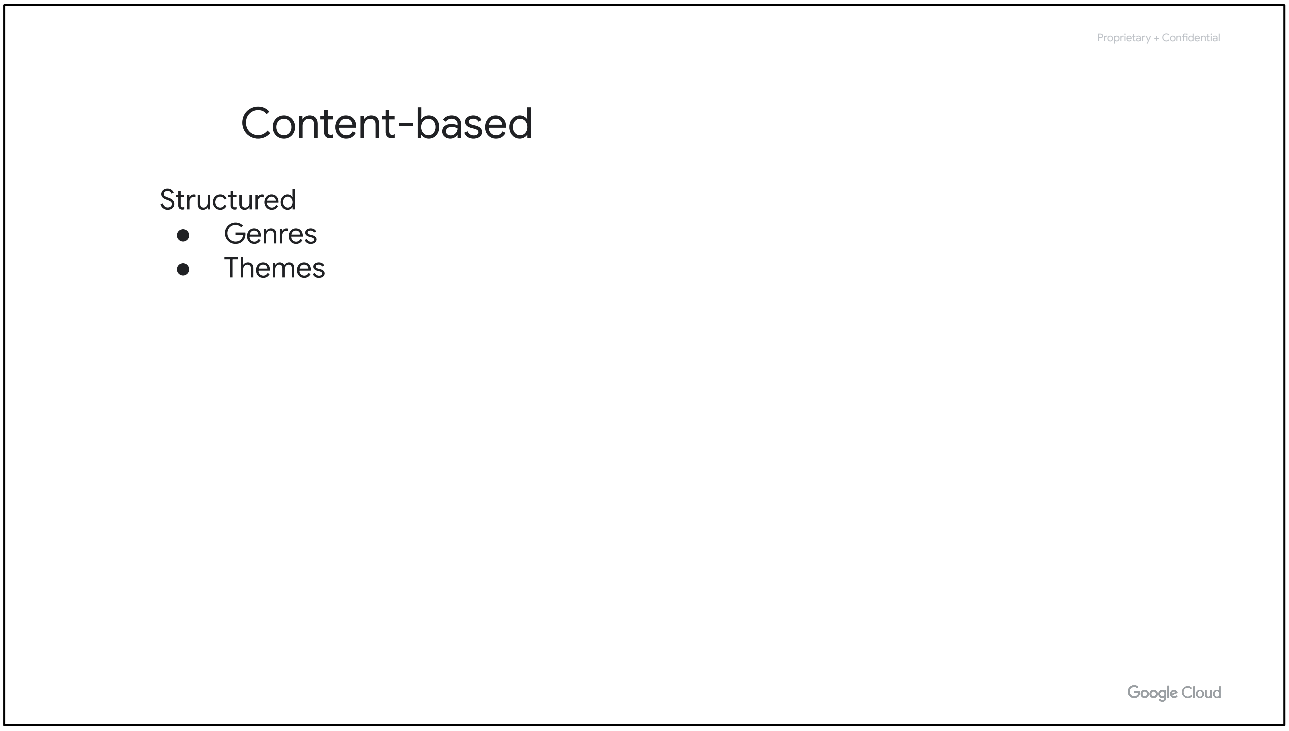
Content-based

Structured

- Genres

Google Cloud

A simple structured piece of data on a movie's page could be the genre of the movie. Note that there could be multiple genres labeled per movie, so we might create an n-hot encoding for each genre per movie. Users often enjoy specific genres, therefore if we know the ones they like, we can recommend more of that type. Because this is content-based, we're going to be making safe genre recommendations because we're basing them on each user's personal genre bubble in embedding space and therefore probably won't explore outside of it. Collaborative filtering will help make up for that by looking across other users too.

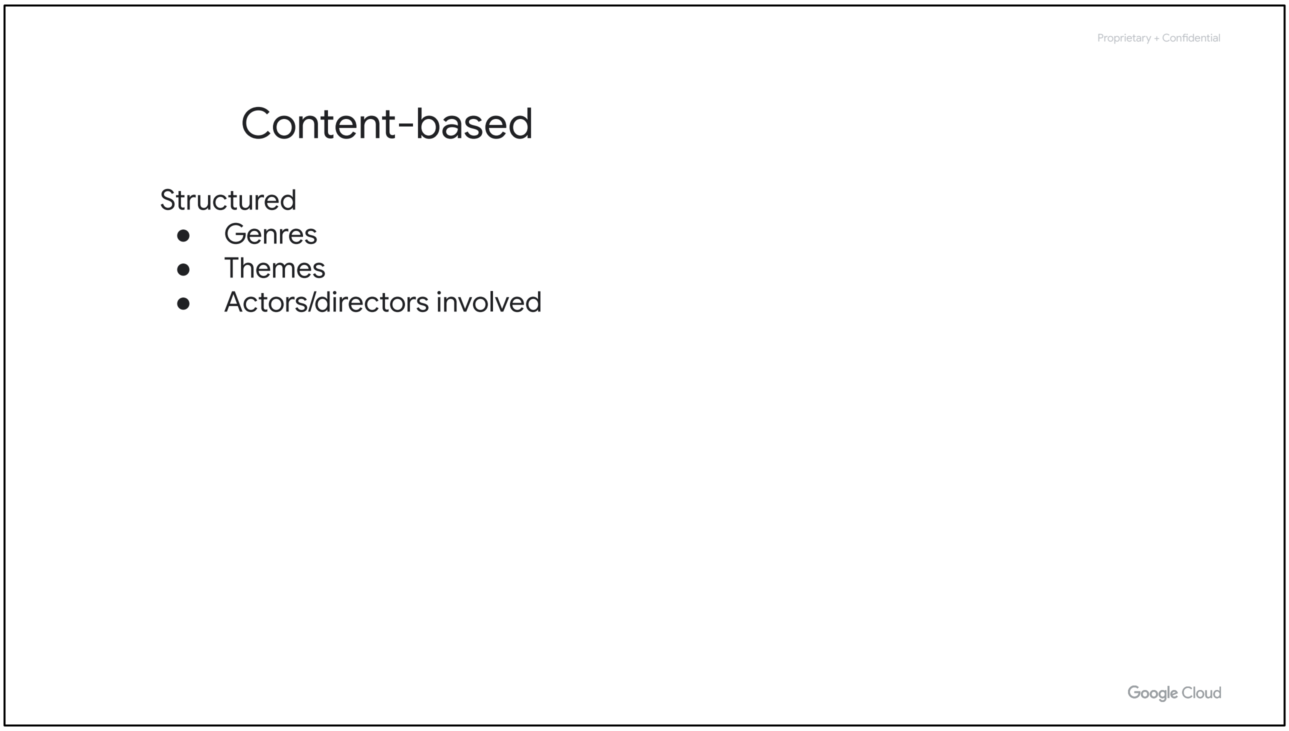

Proprietary + Confidential

Content-based

Structured
- Genres
- Themes

Google Cloud

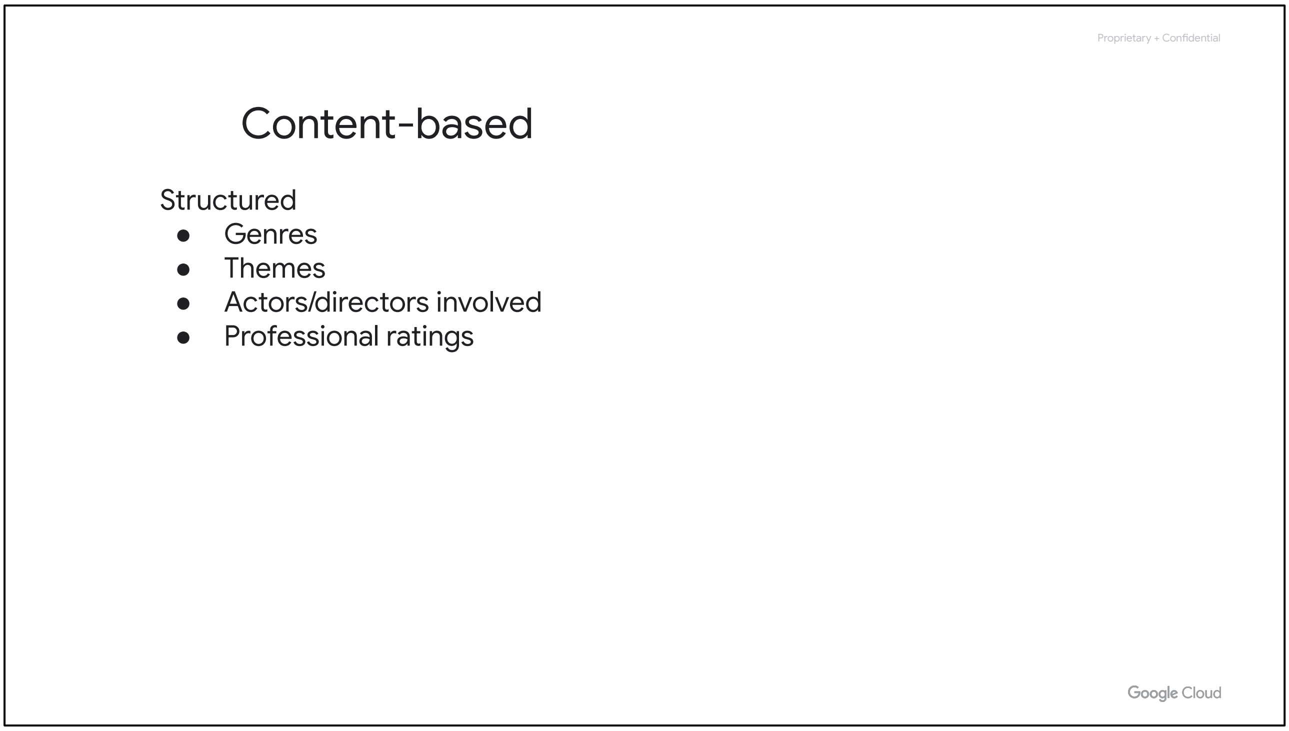
We could also extract the theme of the movie, and just like with genre, there could be more than one, so we should n-hot encode these too. Many people think genres and themes are the same thing, or at least very related, but the cross between the two does create some noticeable differentiation. Perhaps, a user likes sci-fi and also likes the hero journey, the underdog good guys triumph over the in-power bad guys, and hope themes. Not all sci-fi movies have those themes, and those themes exist in other genres outside of sci-fi; therefore, it can help provide another layer of insight into what other movies to recommend. Also, just as with genres, because this is content-based, the recommendations probably won't be far outside users' thematic comfort zones.



Content-based

Structured
- Genres
- Themes
- Actors/directors involved

Another type of structured data we could use is the actors and directors involved. This also could be n-hot encoded for each actor and director that is attached to the movie. Many people have favorite actors and directors and specifically choose movies based on that, due to their personality, style, etc. For instance, maybe a user watches movies all with the same actor in them, portraying many varied roles across multiple genres. There is a good likelihood that other movies that star that actor will also be enjoyed by the user. Because this is content-based, there won't be many movies recommended that don’t have their favorite people in it, but fortunately because most movies have an ensemble of talent, at least the user will be exposed to them too, and might find they like, or dislike, some other people as well while they watch.



Proprietary + Confidential

Content-based

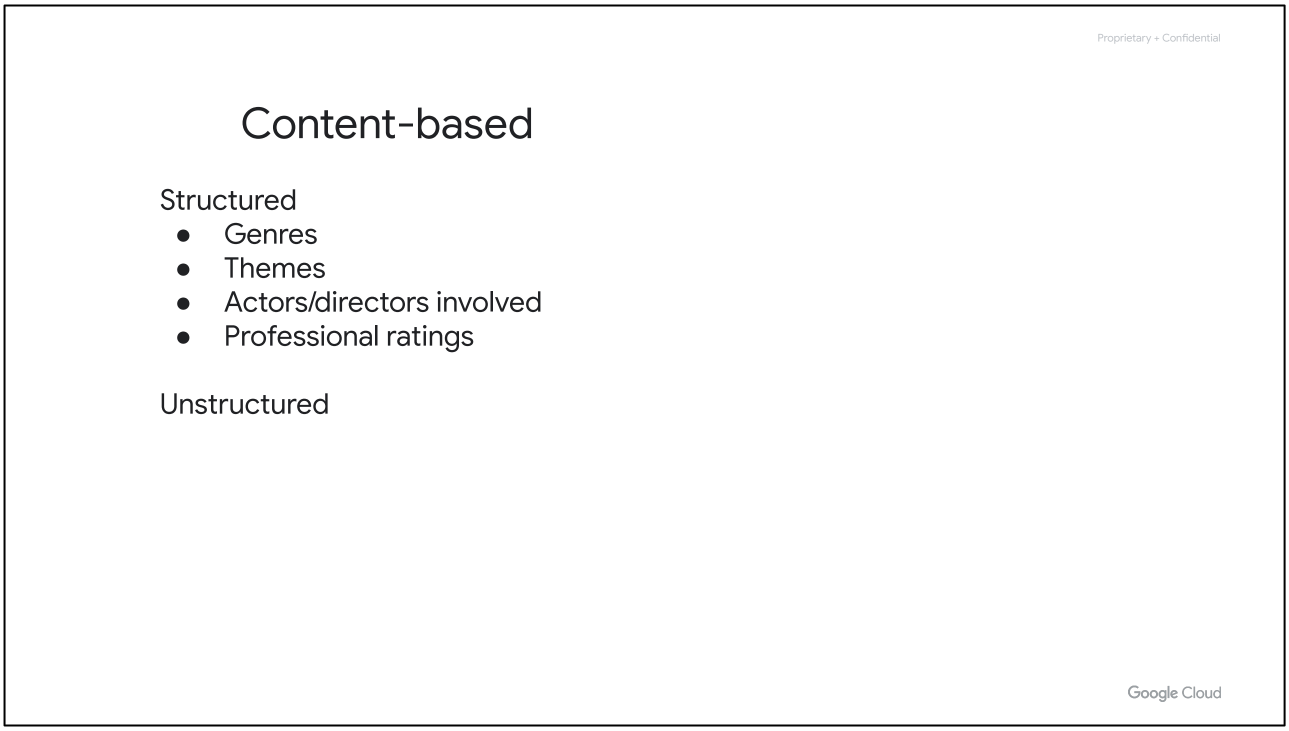
Structured

- Genres
- Themes
- Actors/directors involved
- Professional ratings

Google Cloud

We could have also come up with the idea to tap into professional ratings from critics and paid reviewers. But wait just one moment? Aren't ratings supposed to be for collaborative filtering? Well yes, but those are ratings from users interacting with items, so that we can use both the similarity between items and users to find the best recommendations. In this case, however, these professionals are not users, or at least they are not posting from their personal user accounts and probably have a separate professional opinion area that they enter their ratings in.

It is interesting to think about how these can be used. Perhaps a user usually likes movies that the critics rate highly, whereas other users might actually enjoy movies that get low ratings from the mainstream critics, and everything in between. Because this is content-based, you will likely just get more of the same, professional ratings-wise at least.

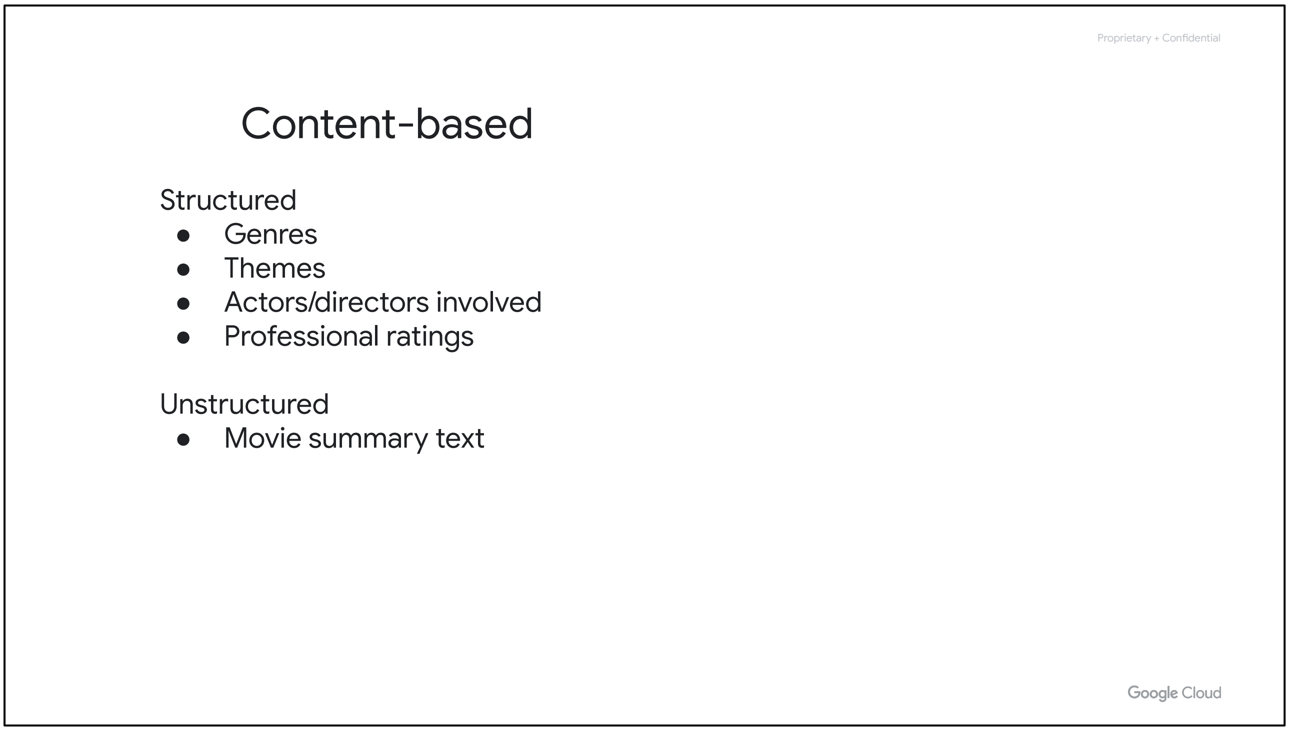


Content-based

Structured
- Genres
- Themes
- Actors/directors involved
- Professional ratings

Unstructured

There's also a massive amount of unstructured data that we could use from the movie webpage.

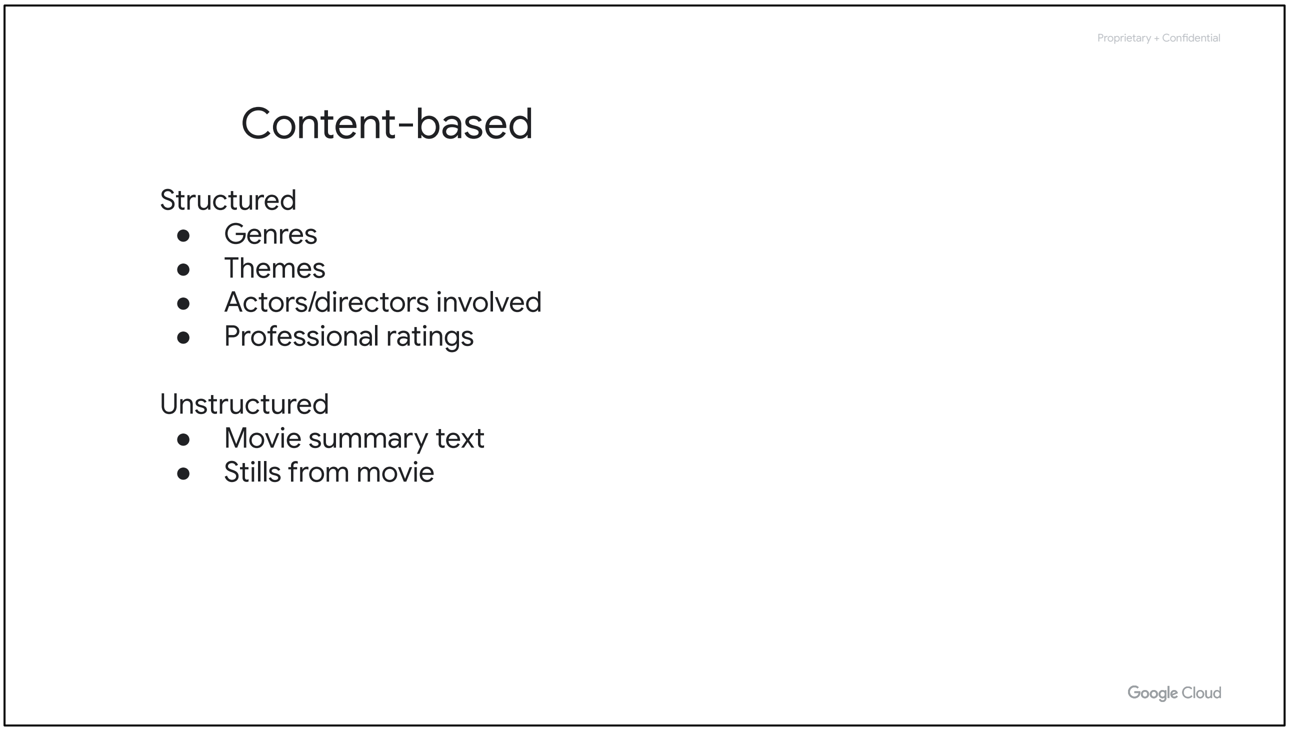


Content-based

Structured
- Genres
- Themes
- Actors/directors involved
- Professional ratings

Unstructured
- Movie summary text

One example could be taking the movie description, synopsis, and full plot summary and performing natural language processing on it to develop a meaningful embedding. This could get at some more of the nuance of what people enjoy about a movie, in addition to boxed-in labels such as genre and theme. If all of the movies a user has watched contain a bunch of references to car chases in the plot summary, they probably enjoy them or at least movies that contain them; therefore, because this is content-based, by using this data we could provide recommendations that are more likely to have car chases.

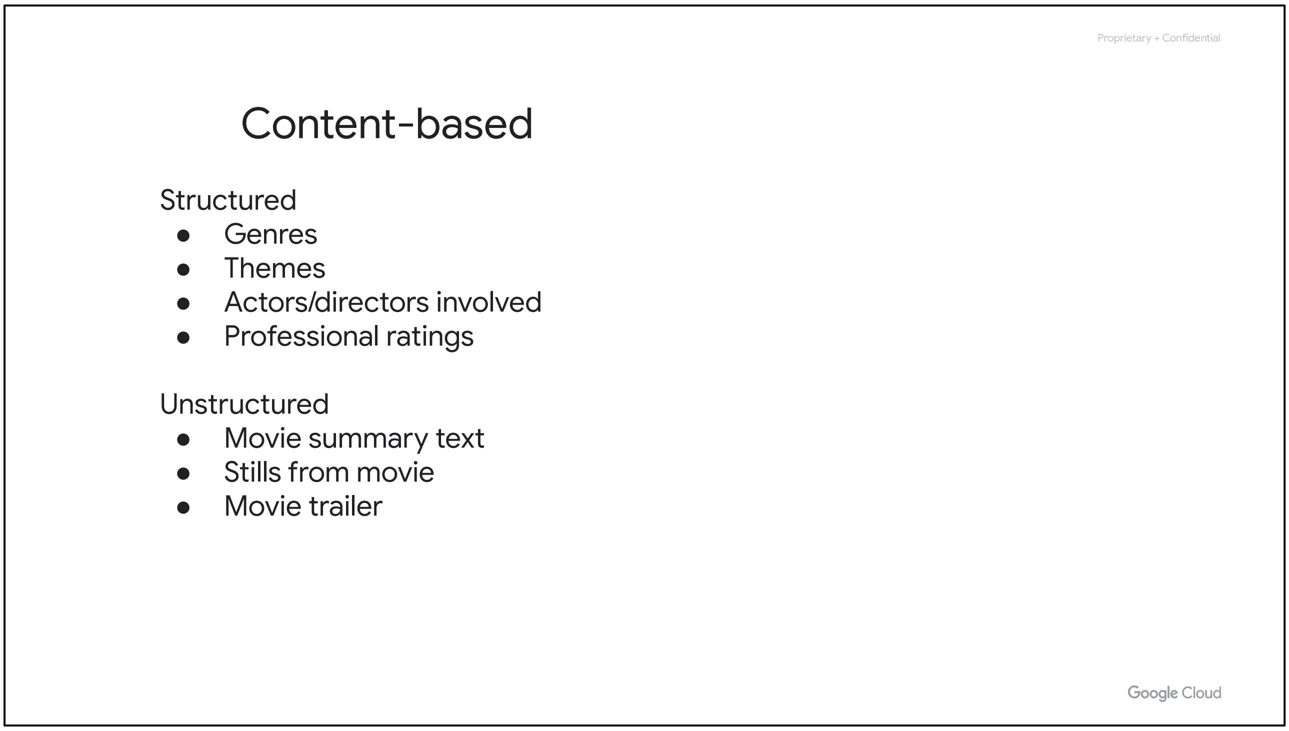


Content-based

Structured
- Genres
- Themes
- Actors/directors involved
- Professional ratings

Unstructured
- Movie summary text
- Stills from movie

In addition to using the text, we can also use promotional images and stills from the movie. This is yet another route to hopefully understand what users like about a movie. We'd have to process the images first through an image model to create some structure. For instance, perhaps a user likes movies that happen to have images of action scenes, and another user likes movies that happen to have a lot of images of space scenes. We can use those to recommend more of the same, but this time based on images attached to the movies.

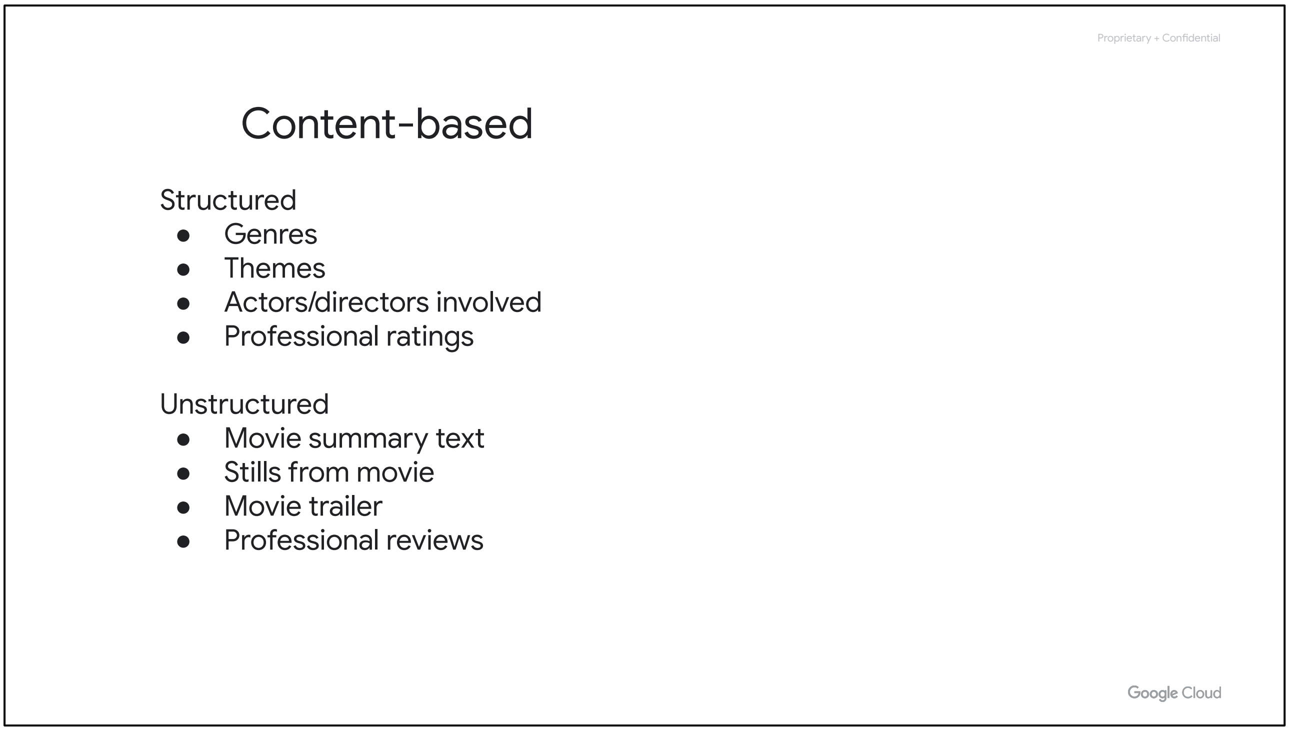


Content-based

Structured
- Genres
- Themes
- Actors/directors involved
- Professional ratings

Unstructured
- Movie summary text
- Stills from movie
- Movie trailer

Why just do text and image, when the great thing about movies is they are quite literally video, which we can also use in a content-based model. Conveniently, most movies these days have multiple trailers, teaser trailers, and commercials attached to them. We can extract the audio, frames, and video-evel information and use that to compare across other movies to recommend similar ones to users. Perhaps the audio of the movies a user likes contain a lot of laughing, so maybe they enjoy comedies, or maybe there is a lot of screaming and running, so they may enjoy horror. The same can be done with the video of chase scenes, long dialogue scenes, nature scenes, hospital scenes, etc.



Content-based

Structured
- Genres
- Themes
- Actors/directors involved
- Professional ratings

Unstructured
- Movie summary text
- Stills from movie
- Movie trailer
- Professional reviews

Lastly, for content-based, we might have thought about using professional reviews posted on the movie's webpage. These, unlike the professional ratings, are unstructured and first need to have natural language processing applied to create some structure, and then we can use these in our content-based model. Remember, as with the professional ratings, these reviews are not written by users, so this is not collaborative filtering. Furthermore, users who like certain movies could agree or disagree with the professional reviews, and because it is content-based, they will be recommended more of the same in that dimension.

Of course we could have thought of many more pieces of data we could use, like language of the movie, etc., but this is enough for now for content-based recommendation systems.

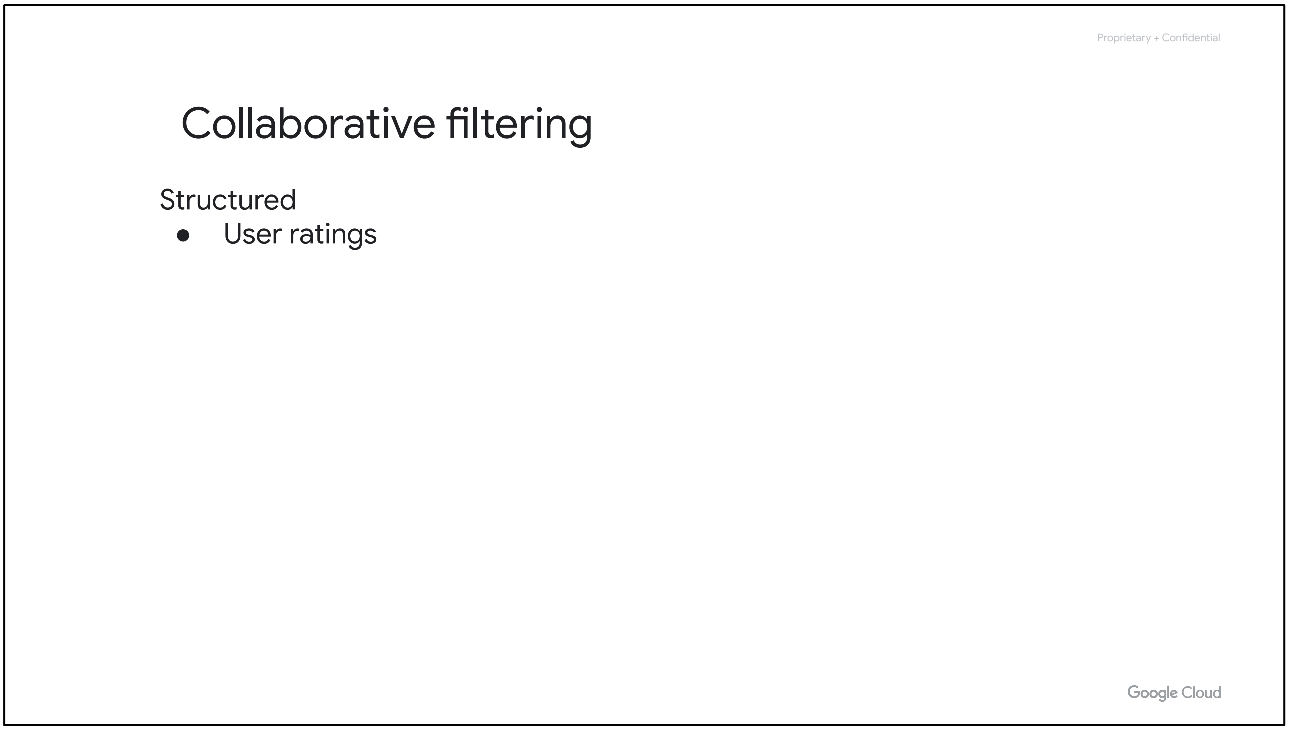
Proprietary + Confidential

Collaborative filtering

Structured

Google Cloud

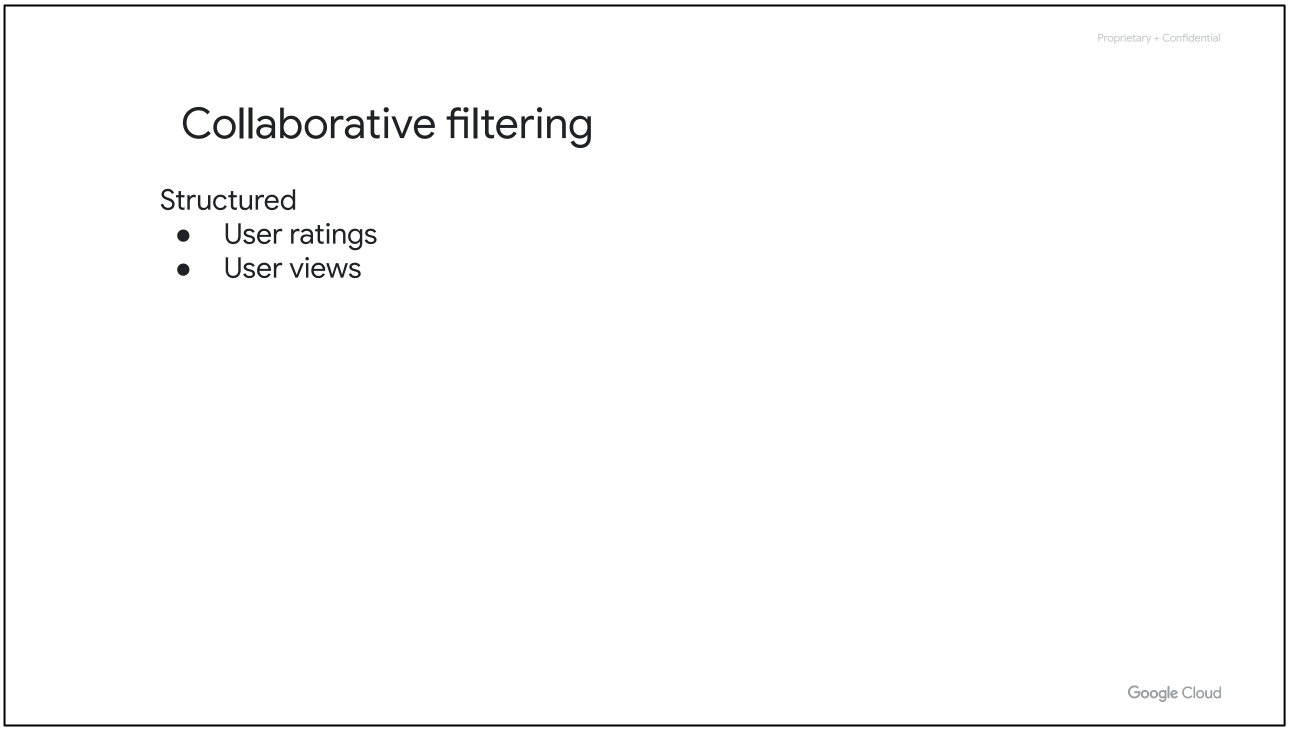
All right, let's see some of the ideas we came up with.



Collaborative filtering

Structured
- User ratings

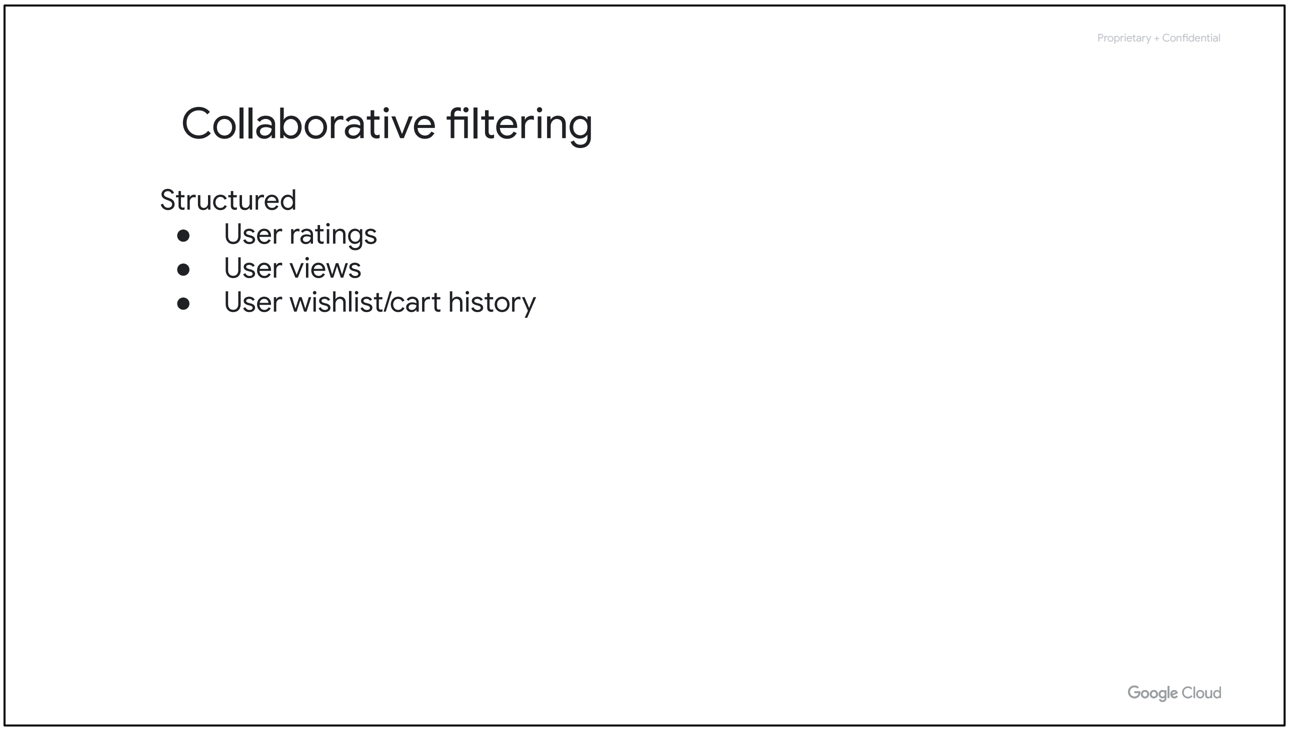
The most obvious thing to use for collaborative filtering is, of course, user ratings. Now this website might not have a way for users to leave explicit ratings, and we would have to resort to more implicit means of feedback, but for this hypothetical example let's assume there are ratings. As we know with collaborative filtering, these ratings will be used to create user and item factors within a rating embedding space of latent features that we can then use to make recommendations.



Collaborative filtering

Structured
- User ratings
- User views

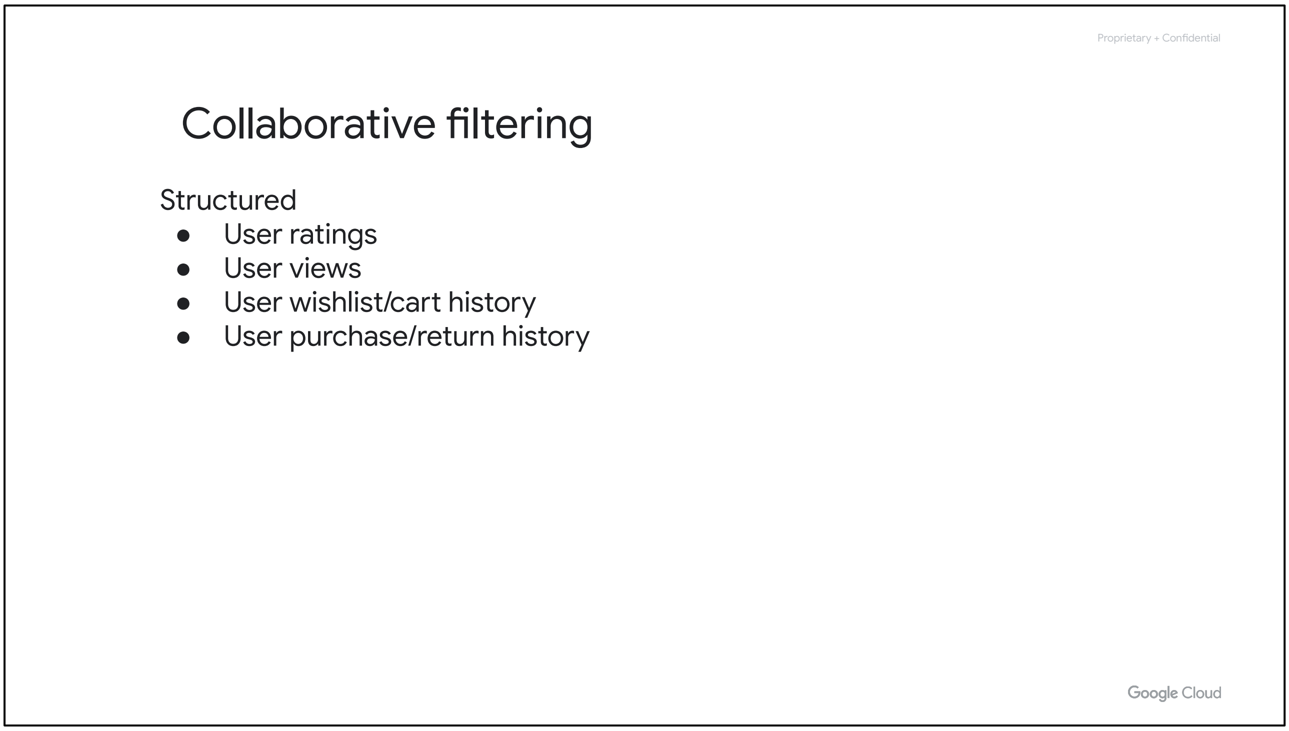
We can also use implicit feedback, such as user views. We can assume that a user ended up on a movie's webpage for a reason; that some interest led them there. Perhaps instead of just a flag if a user has visited a movie's webpage, we can keep track of the number of times they visited the page. This goes with the assumption that if a user visits a movie's page more frequently, they have more interest than for movie webpages that they don't visit frequently or at all.



Collaborative filtering

Structured
- User ratings
- User views
- User wishlist/cart history

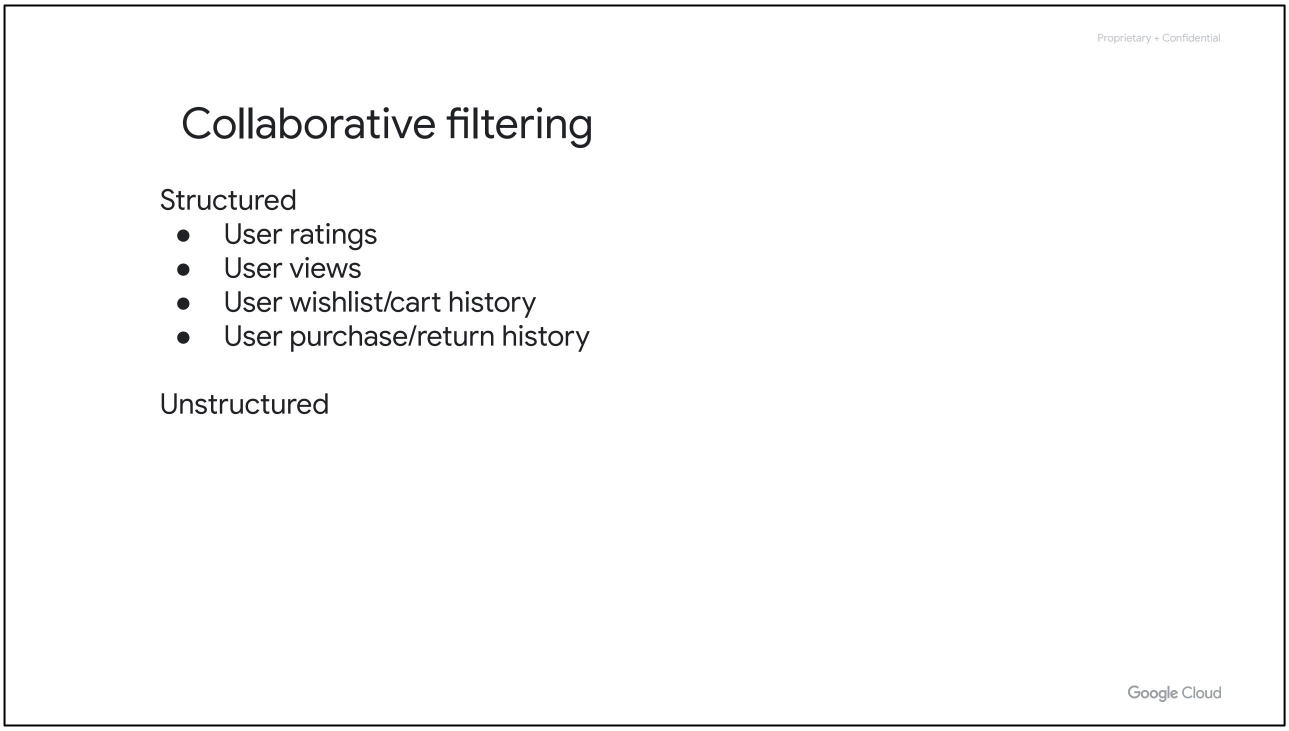
There's also the user wishlist/cart history that we could use. This is an interesting dataset. Thinking about the wishlist, it is implicit because it is not explicitly saying that a user likes the movie, but that they think they might like it enough to add to their list. Also, the cart history is another step further. The user has actually now selected the movie for purchase, and they just need to check out. As a lot of us do, though, putting something into a cart doesn't mean we will buy it, but it is just another interaction, another signal, that this user might like this particular movie.



Collaborative filtering

Structured
- User ratings
- User views
- User wishlist/cart history
- User purchase/return history

Going one step further, we can use the user purchase history as implicit feedback, because we still don't know whether they like or dislike the movie, but it is an interaction. However, we may be able to also use a user's return history as a proxy for dislike, because it is probably more likely that someone will return a movie that they don't like, over potentially other reasons that are more rare, like the movie is broken, wrong language version sent, etc.



Proprietary + Confidential

Collaborative filtering

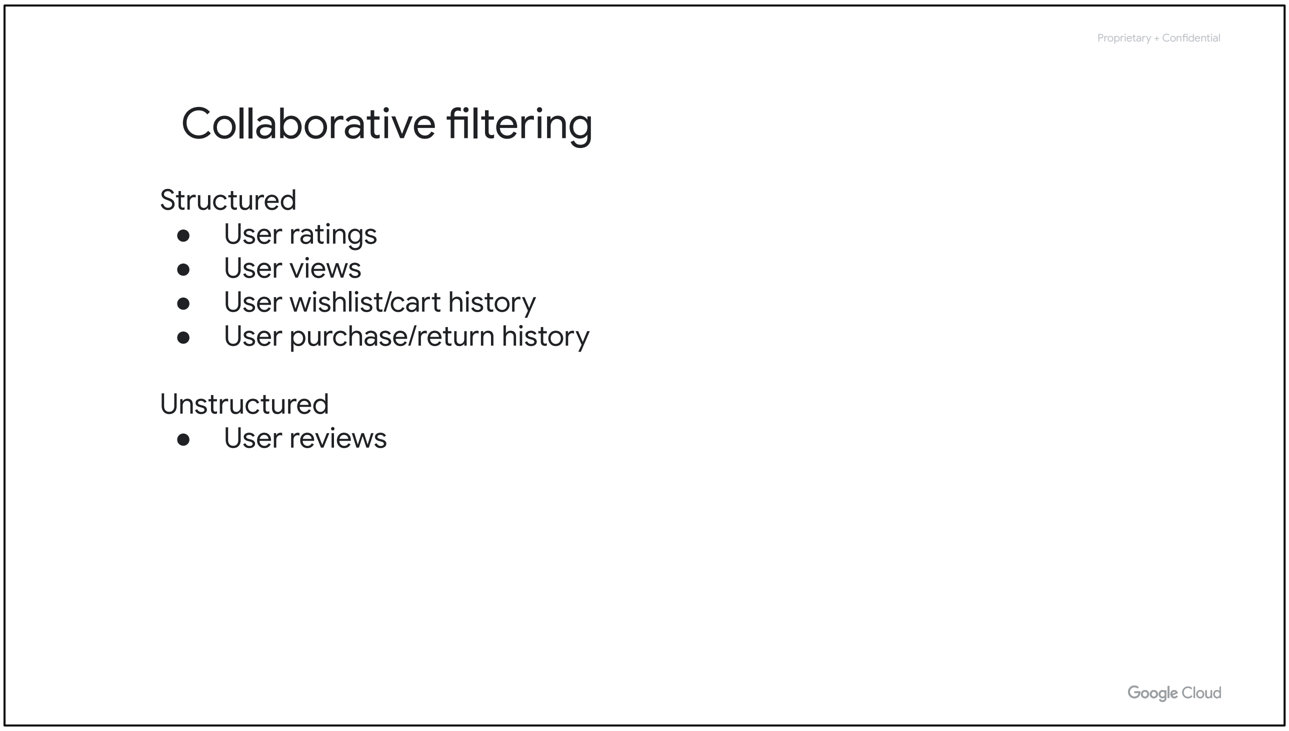
Structured

- User ratings
- User views
- User wishlist/cart history
- User purchase/return history

Unstructured

Google Cloud

There is also unstructured data we can use for our collaborative filtering model:

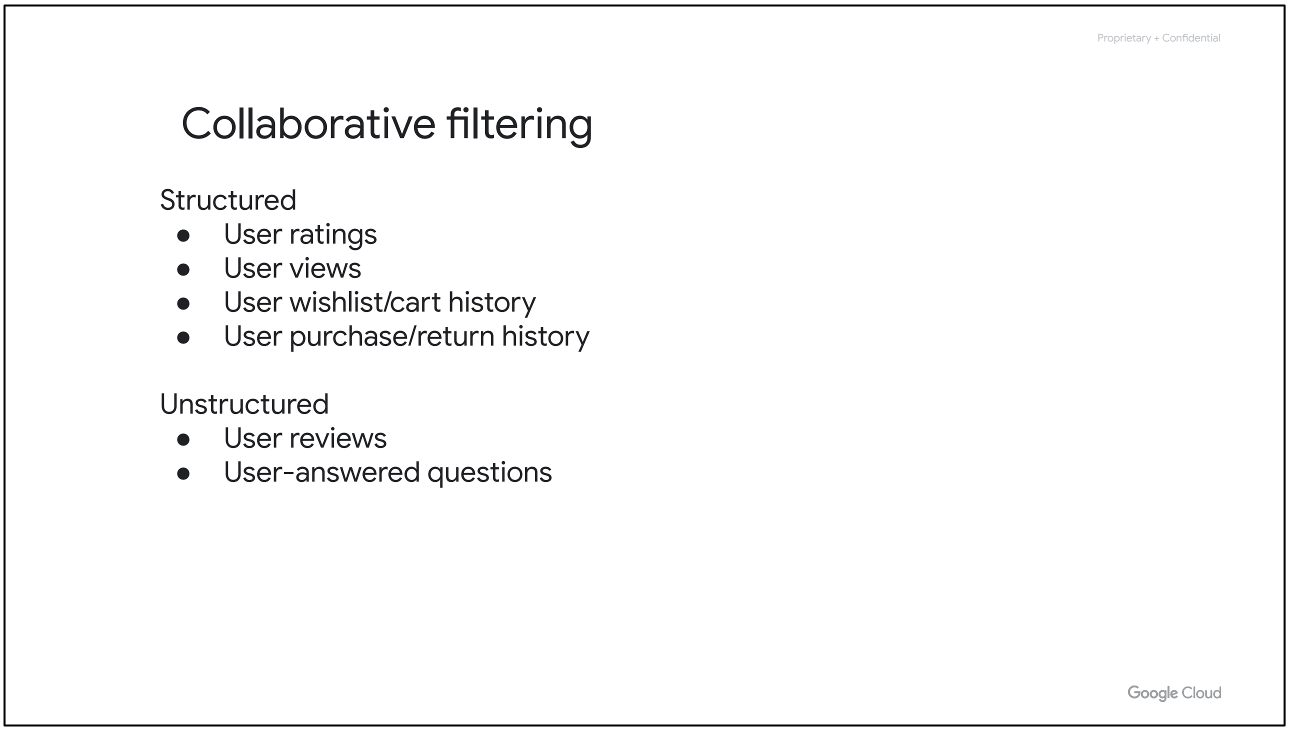


Collaborative filtering

Structured
- User ratings
- User views
- User wishlist/cart history
- User purchase/return history

Unstructured
- User reviews

One type is user reviews. These will be free text that we can apply natural language processing to in order to gain some idea of sentiment. Unlike content-based's professional reviews, these are written by users; users that we can find similarities to in order to make recommendations from.

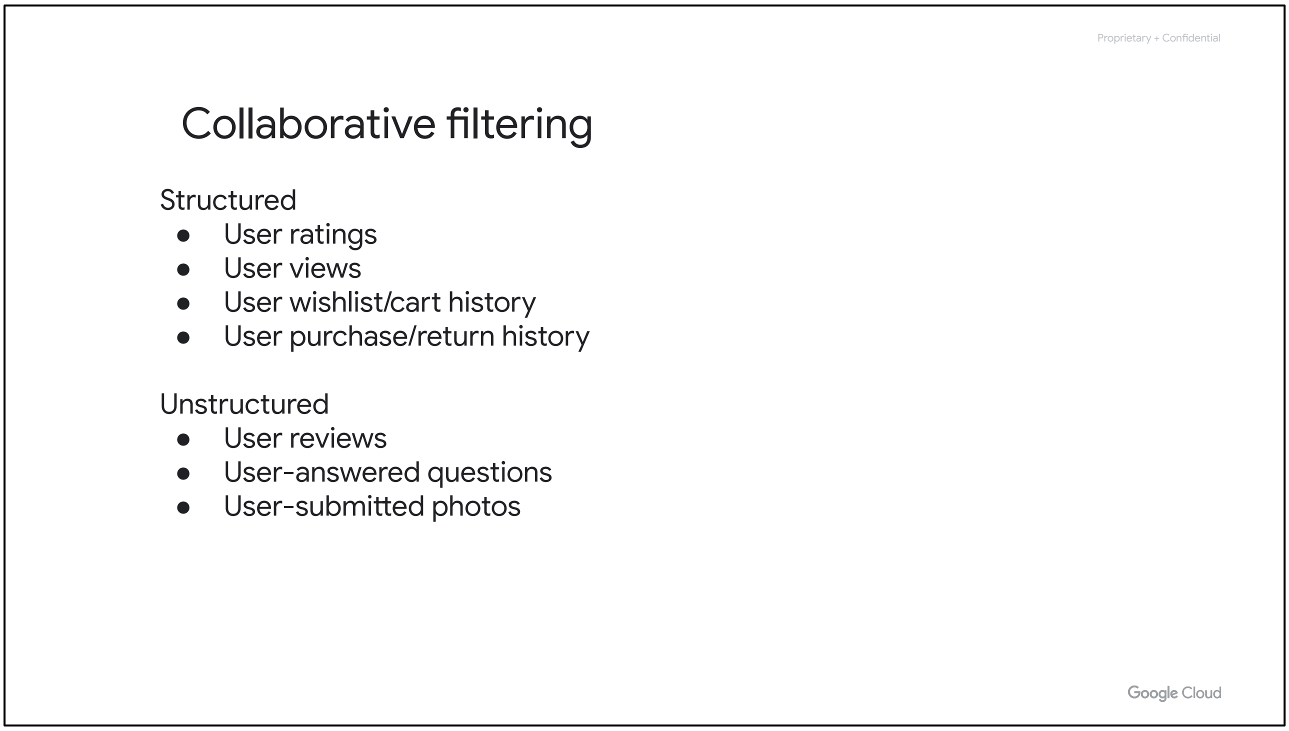


Collaborative filtering

Structured
- User ratings
- User views
- User wishlist/cart history
- User purchase/return history

Unstructured
- User reviews
- User-answered questions

We can also try using user-answered questions as data for our collaborative filtering model. These can be a little bit tricky. A user answering a question doesn't mean that they like the movie. In fact their answer to a question could indicate that they don't like the movie. So we could just use a yes or no flag or the number of questions answered for each user for that movie as implicit feedback, because answering is an interaction, similar to using user views. We could also run sentiment analysis on the answers, and although most would probably be flagged as neural, some may show as positive and others negative, which might be enough extra signal to improve our recommendations.



Collaborative filtering

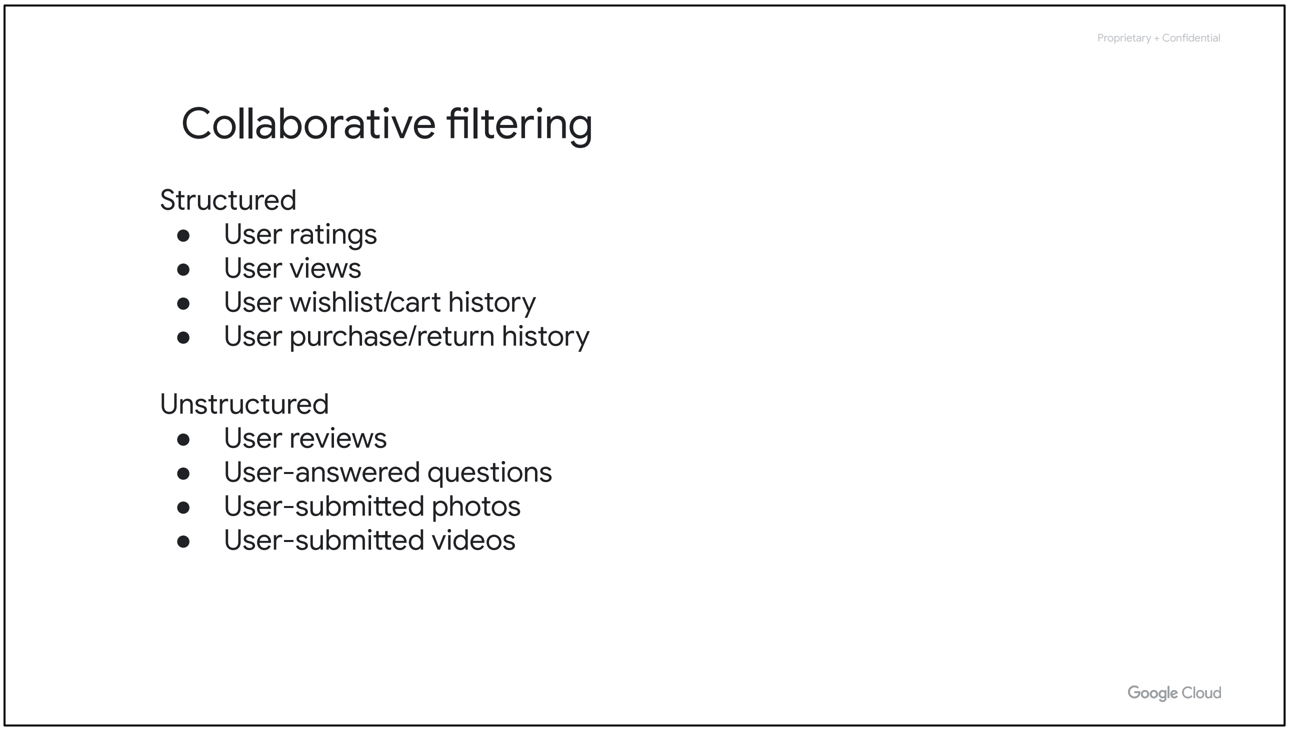
Structured

- User ratings
- User views
- User wishlist/cart history
- User purchase/return history

Unstructured

- User reviews
- User-answered questions
- User-submitted photos

There could also be user-submitted photos for some movies. This might apply more to people buying items and submitting pictures of the item all assembled etc. However, perhaps people show how the movie looks on their brand new TV, or they take pictures of themselves at a movie-viewing party with their friends in their home theater. All of these scenarios are data that may, or may not, add some signal and help improve recommendations. We can use the fact that someone uploaded a photo at all, or the number of photos they uploaded, as a form of implicit feedback. We could also use an image model to create trained labels that might convey some form of sentiment.

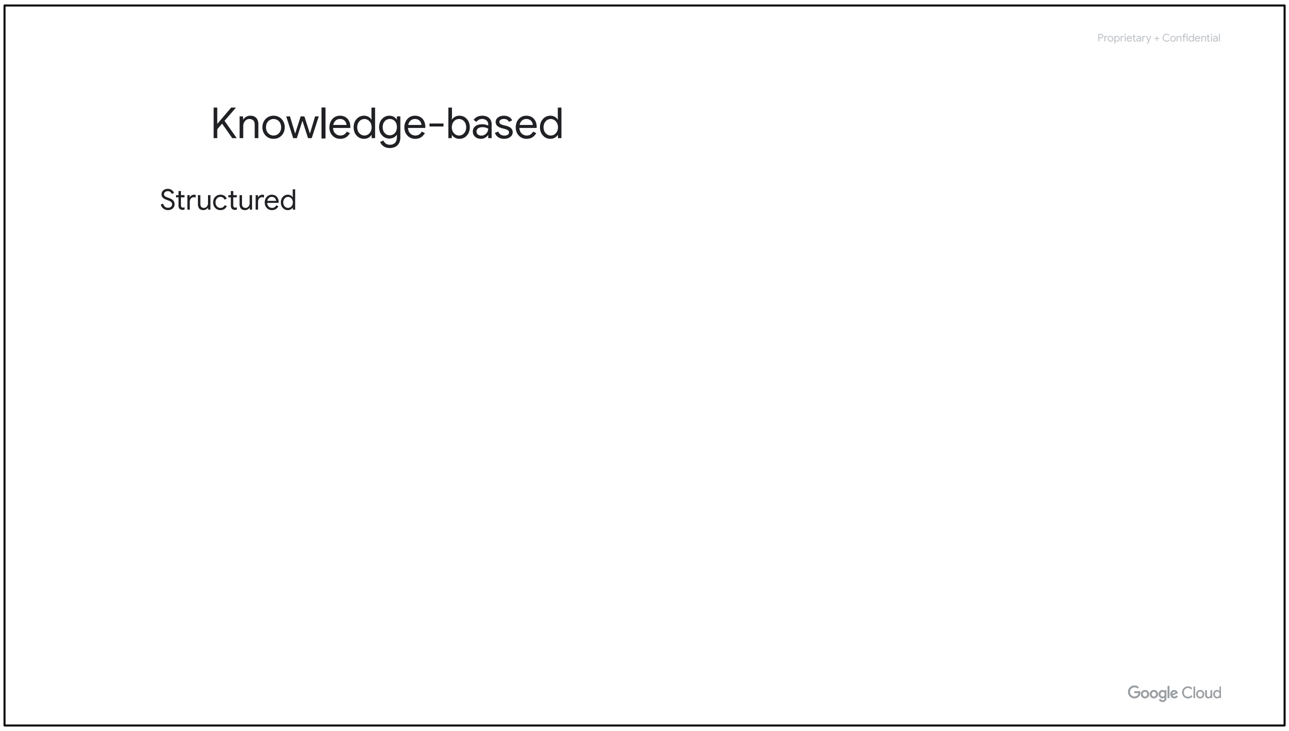


Collaborative filtering

Structured
- User ratings
- User views
- User wishlist/cart history
- User purchase/return history

Unstructured
- User reviews
- User-answered questions
- User-submitted photos
- User-submitted videos

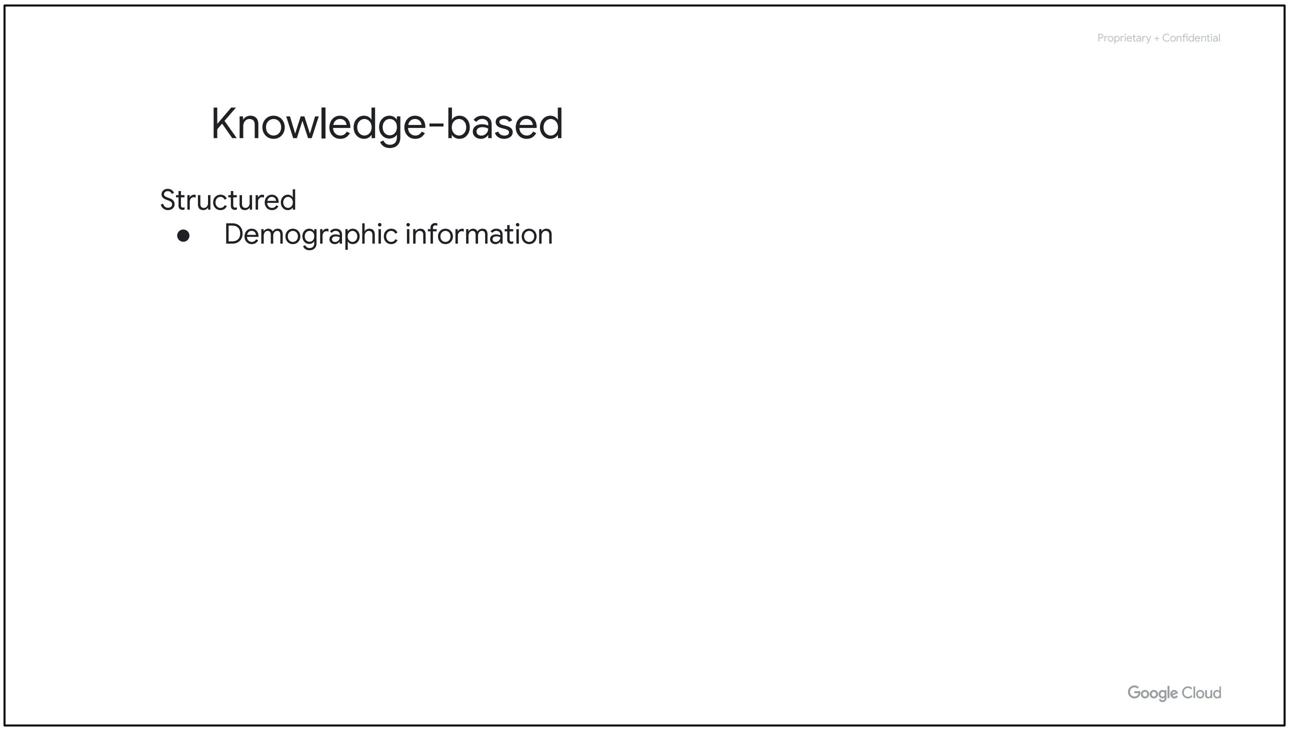
The same goes for user-submitted videos, where we can use the fact that they interacted as implicit feedback and/or can process the video with other machine learning algorithms and use that now-structured interaction data.



Knowledge-based

Structured

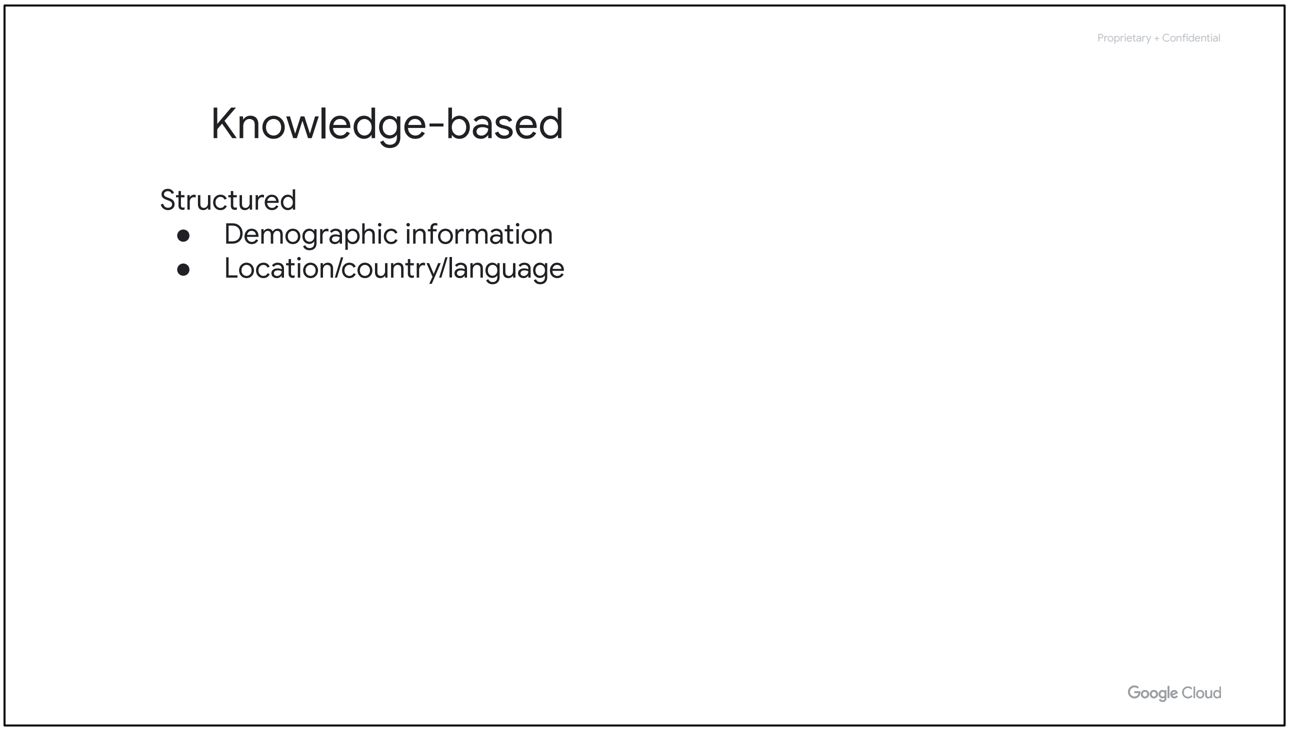
All right, let's see some of the ideas we came up with.



Knowledge-based

Structured
- Demographic information

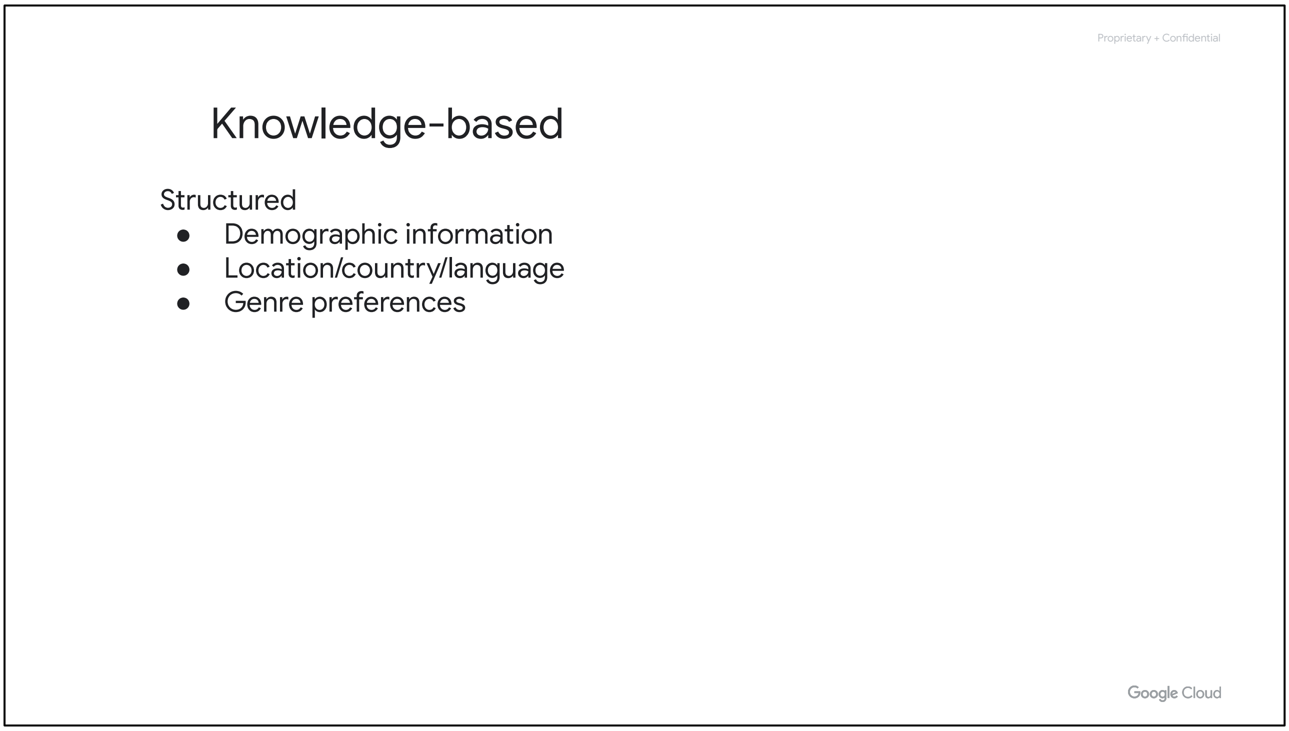
The most obvious user data we could use for our knowledge-based recommendation system is probably demographic information. This could be things like age, gender, etc. Basically any dimensions that could possibly have enough variance to differentiate users from each other, so that we don't just recommend the same movies to everyone. What point of a model is that?! These would probably be structured and bucketed into categories. For example, people with a very young age, in other words children, probably enjoy watching what other children watch, and not something an adult might like. There may be exceptions to these patterns, but that is just what happens in machine learning. We need to be careful though, that we aren't stereotyping and always recommending certain movies to certain demographics. Including other data dimensions should help balance this out, especially if demographics aren't very predictive and the other features are.



Knowledge-based

Structured
- Demographic information
- Location/country/language

Another possible dataset we could use for our knowledge-based recommendation systems is user locations, countries, or languages. This could be important for localization. We could also use language as a feature because if a user doesn't speak Spanish, they may not enjoy watching a Spanish movie, even if it has subtitles. But, as with all machine learning, we need to be mindful that we don't go too far. Perhaps someone who speaks English and lives in the United States learned French at university and really enjoys French movies. Therefore, it is always good to have other features and let the model learn what features are important and which aren't and to balance itself out of such situations.

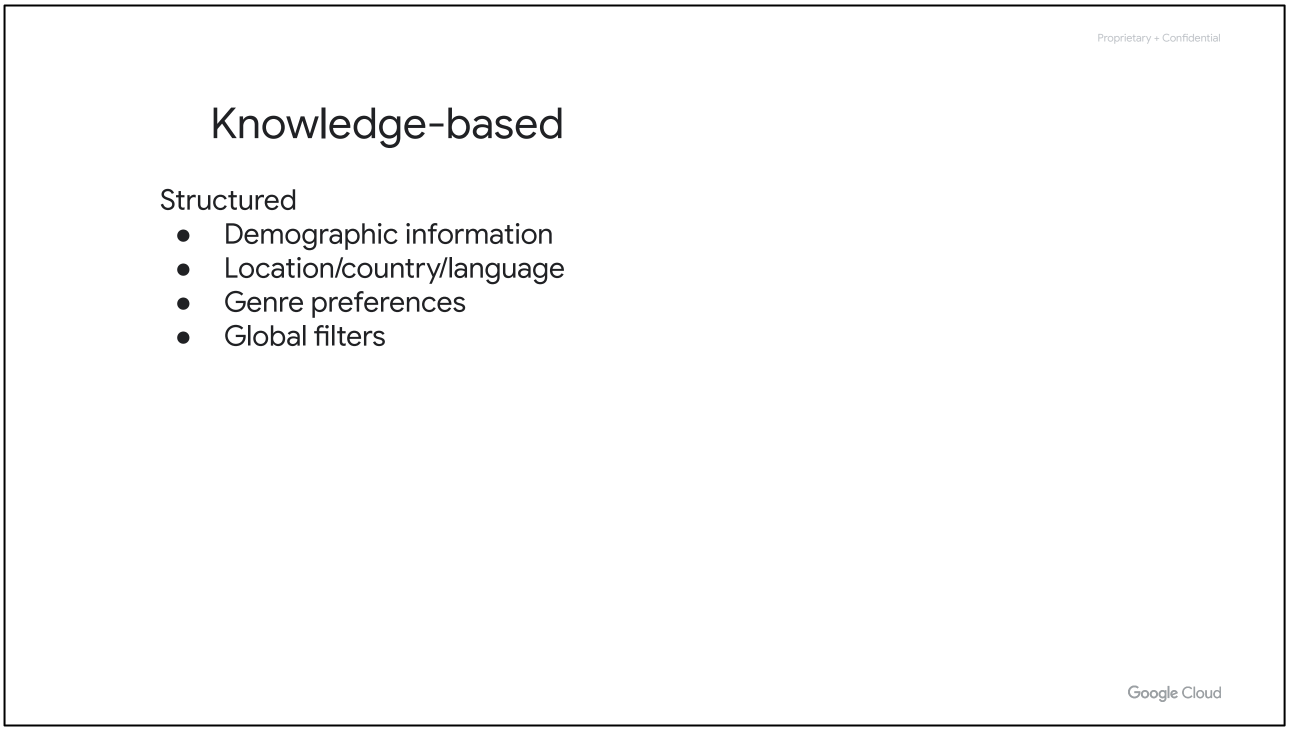


Knowledge-based

Structured

- Demographic information
- Location/country/language
- Genre preferences

Another possibly great feature we could use to try to improve our knowledge-based recommendation system is user-entered genre preferences. This way we know what users feel are important to them genre-wise, and we can use that to recommend the genres they have listed as their favorite. Because this is knowledge-based only, for now, they will only get movies of the genres they have listed, but if paired with collaborative filtering, they might receive some more serendipitous recommendations.

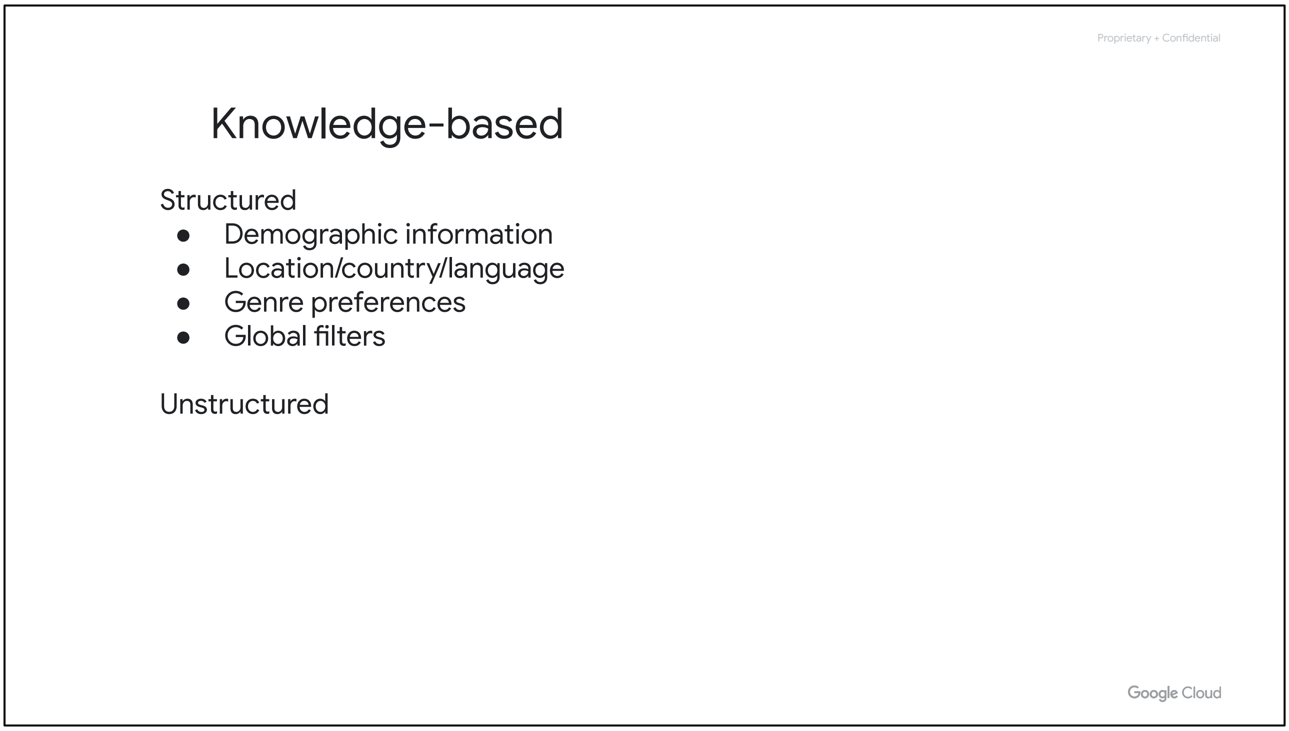

Proprietary + Confidential

Knowledge-based

Structured
- Demographic information
- Location/country/language
- Genre preferences
- Global filters

Google Cloud

Furthermore, building off of that, global filters set by users can be used as another dataset for our knowledge-based recommendation system. These somewhat take the genre preferences a step further, because when we choose to filter things out, we are choosing for some things to remain. For instance, beyond just wanting only certain genres and filtering out the others, we could apply other user-chosen filters such as they only want movies with an average rating of 4 or higher, or only movies that have been nominated or have won an Academy Award. This is essentially just keyword logic to apply these filters, but they do do the job, because you won't be recommended what you've filtered out.

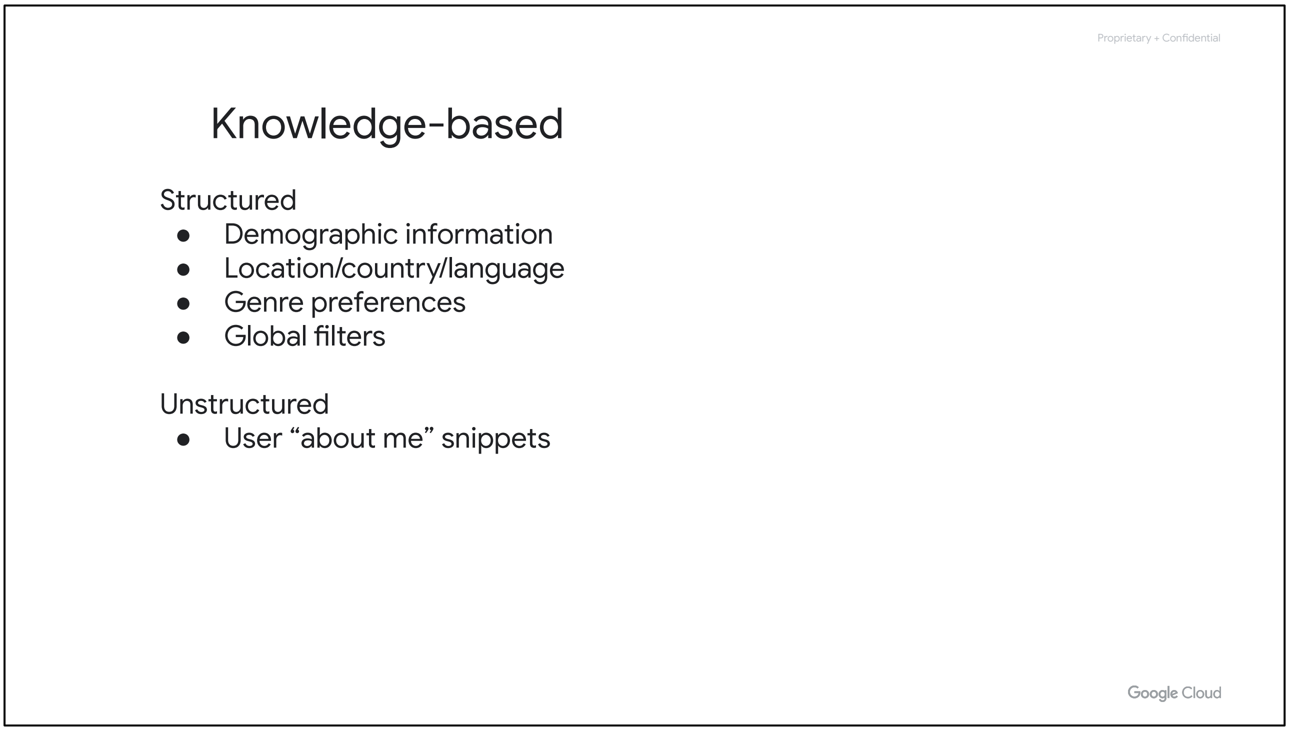


Knowledge-based

Structured
- Demographic information
- Location/country/language
- Genre preferences
- Global filters

Unstructured

As for unstructured data,

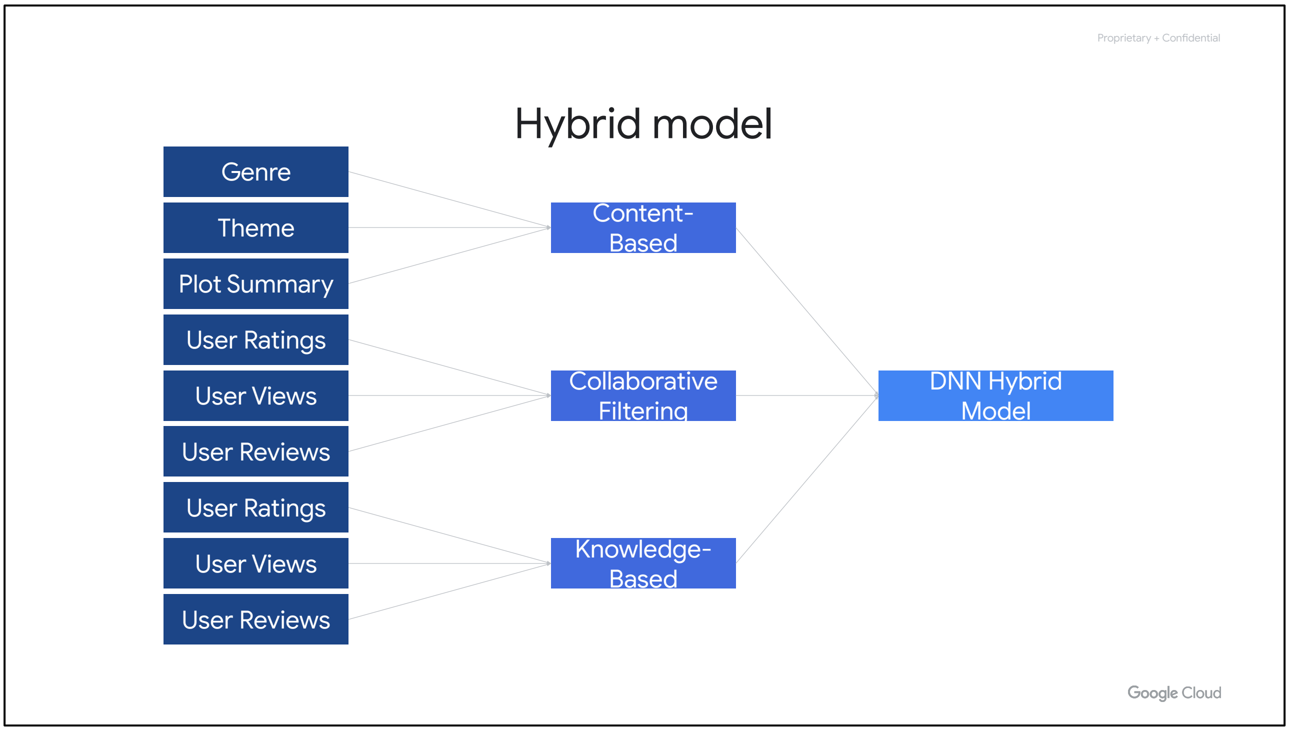


Knowledge-based

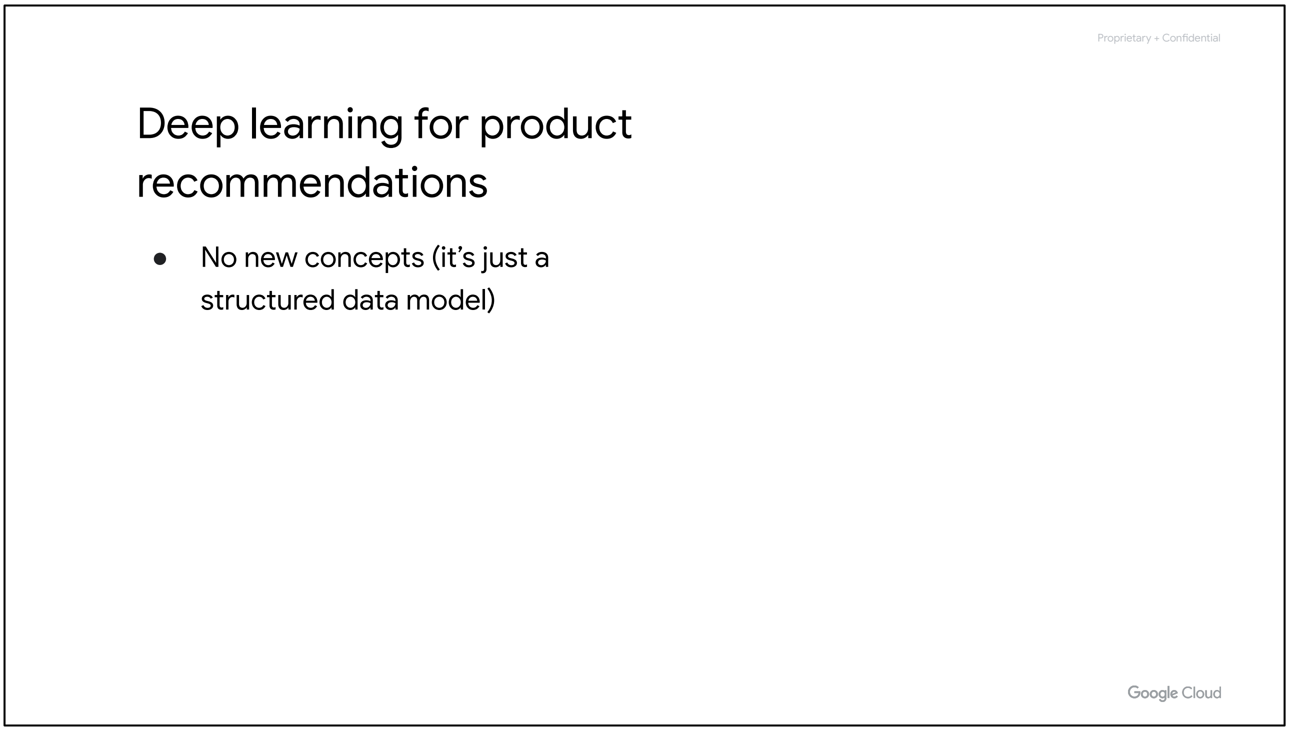
Structured
- Demographic information
- Location/country/language
- Genre preferences
- Global filters

Unstructured
- User “about me” snippets

our website may have an “about me” area for users to enter free text about themselves and their interests in movies. We could apply natural language processing to the text and then compare those word embeddings against other users' word embeddings. However, there could be sparsity problems if not enough users fill out their “about me” page.



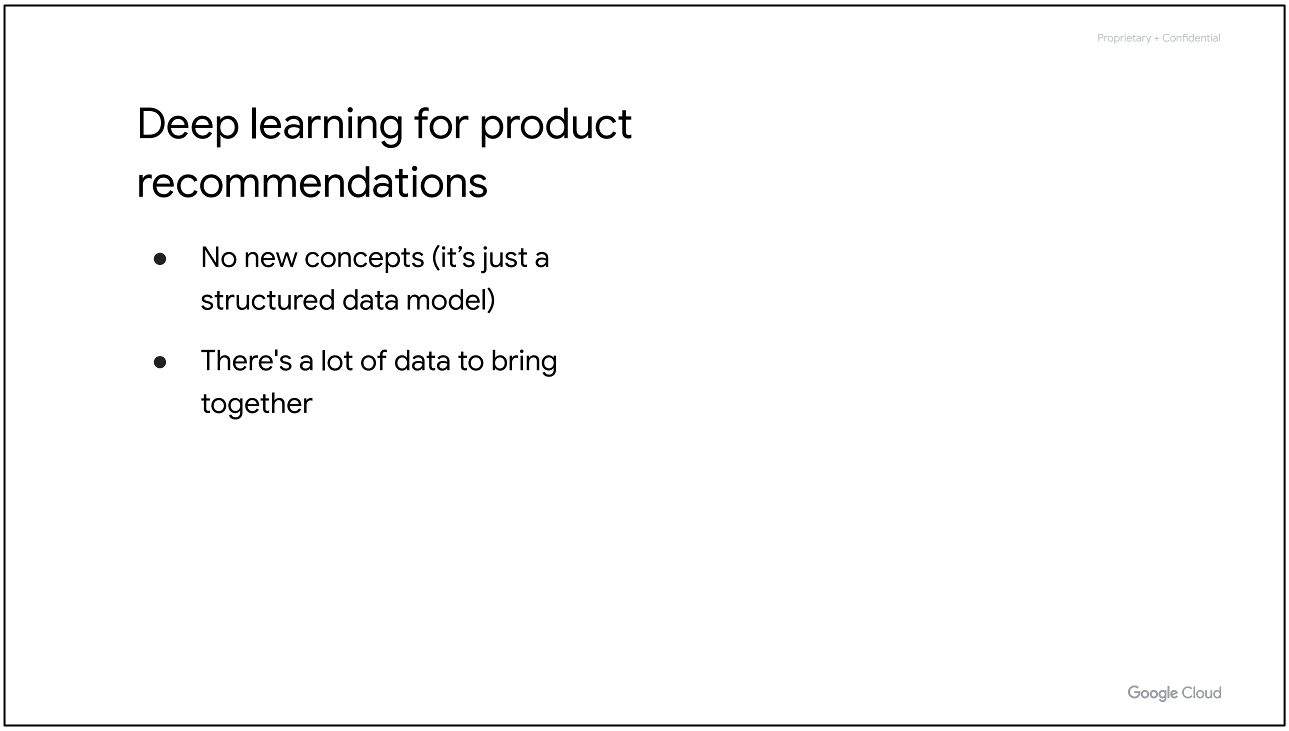
So we've gone through over 20 possible datasets, using the three broad forms of recommendation systems. We went over some of the possible issues that could arise if we were to only look at each dataset's model independently. Fortunately, these issues tend to go away if in concert with other datasets, using the other types of recommendation systems. This leads us to the hybrid model. You can think of each independent dataset model as being a feature to our hybrid model, connected via a deep neural network. This can take the embeddings from collaborative filters and combine them with content information and knowledge about the user.



Deep learning for product recommendations

- No new concepts (it's just a structured data model)

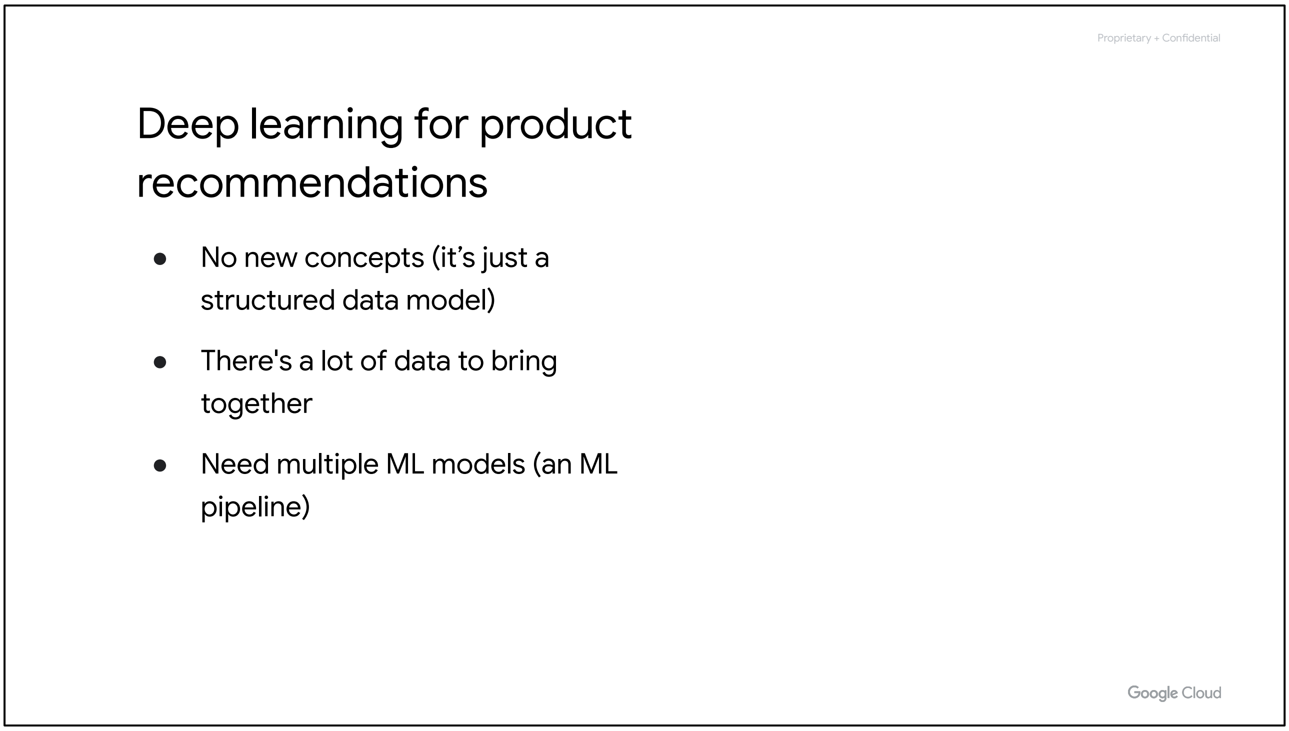
Deep learning for product recommendations is a great way to build upon a traditional, standalone recommendation system. There are no new concepts that we need to learn or implement; we can just use the building blocks we already made. It's just a structured data model, which we have had lots of practice working with already.



Deep learning for product recommendations

- No new concepts (it's just a structured data model)
- There's a lot of data to bring together

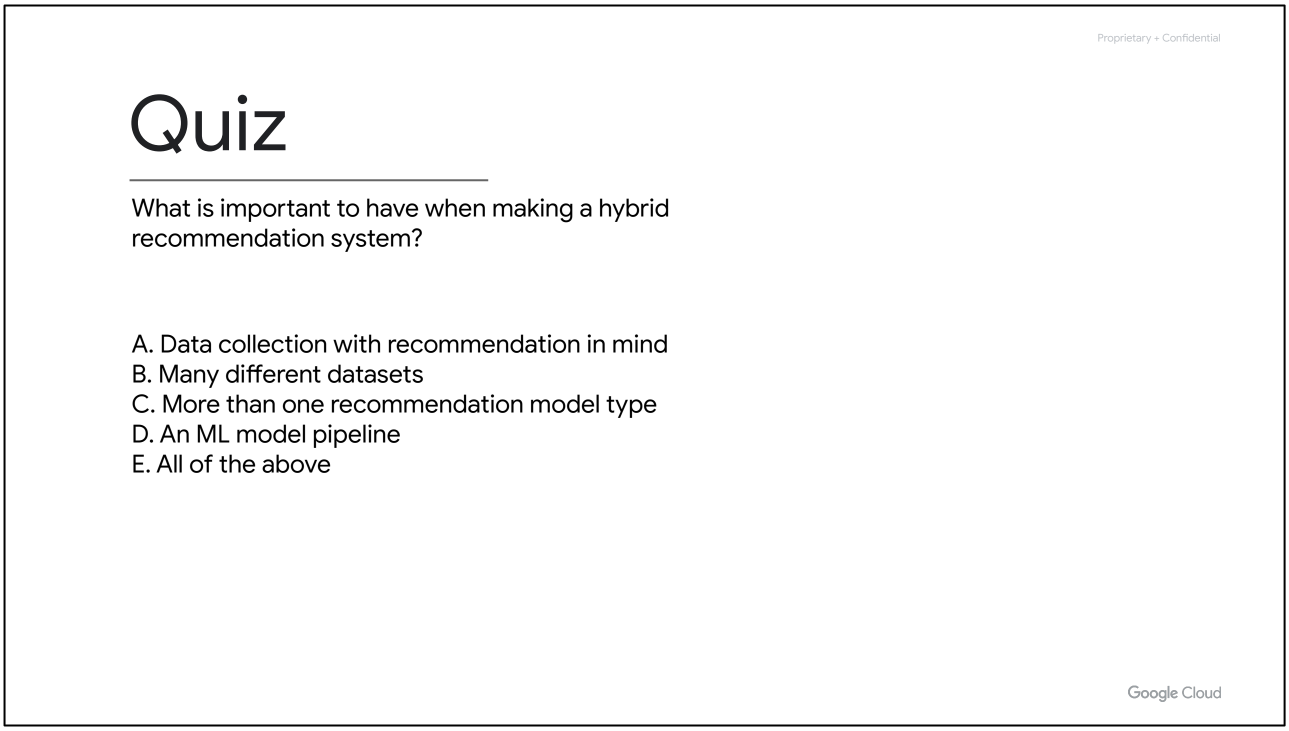
As we saw from the design lab, there's a lot of data to bring together, but this is a good thing because data is the fuel for machine learning. The more ways we can figure out to take advantage of it, the more likely we will have a powerful final model.



Deep learning for product recommendations

- No new concepts (it’s just a structured data model)
- There's a lot of data to bring together
- Need multiple ML models (an ML pipeline)

Therefore, we will need multiple machine learning models connected together in a machine learning pipeline to make the greatest impact.


Proprietary + Confidential

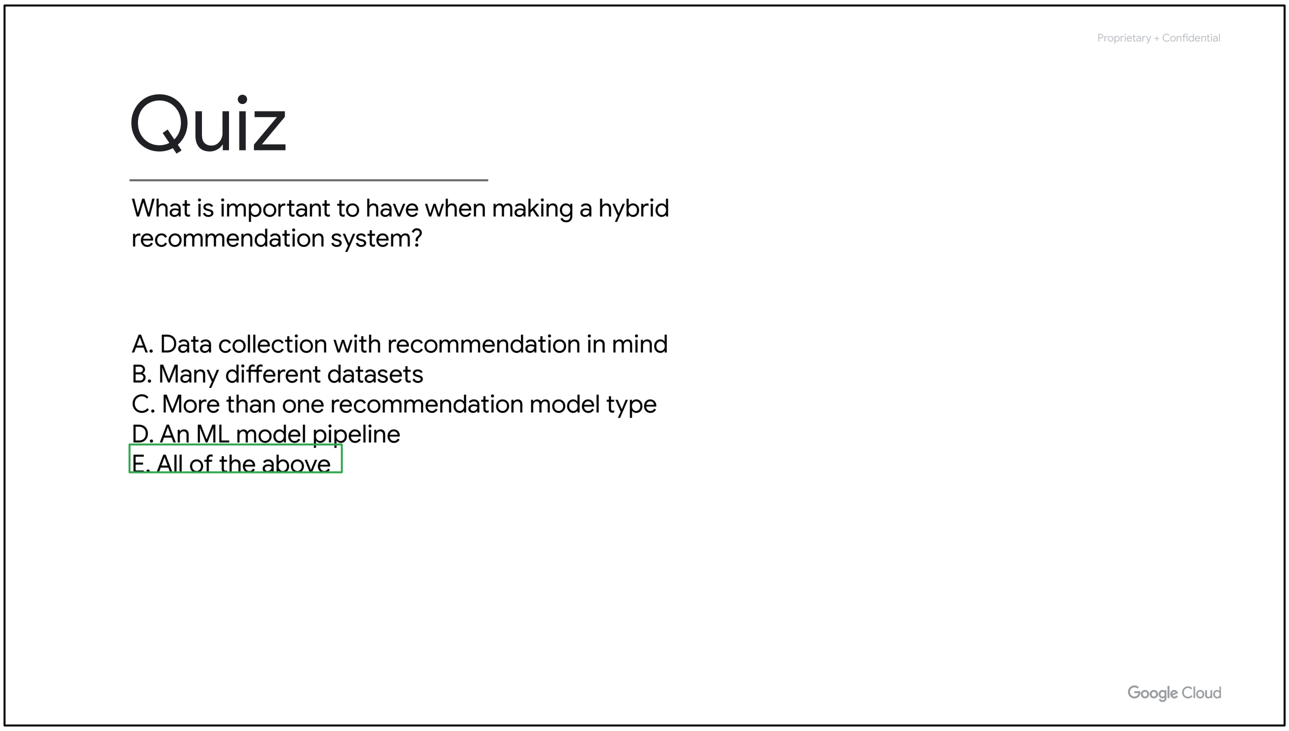
Quiz

What is important to have when making a hybrid recommendation system?

A. Data collection with recommendation in mind
B. Many different datasets
C. More than one recommendation model type
D. An ML model pipeline
E. All of the above

Google Cloud

Let's now test our knowledge about hybrid recommendation systems! What is important to have when making a hybrid recommendation system? Creating your data collection with recommendation systems in mind? Having many different datasets from different sources? Having more than one recommendation model type? Having a machine learning model pipeline? Or all of the above?

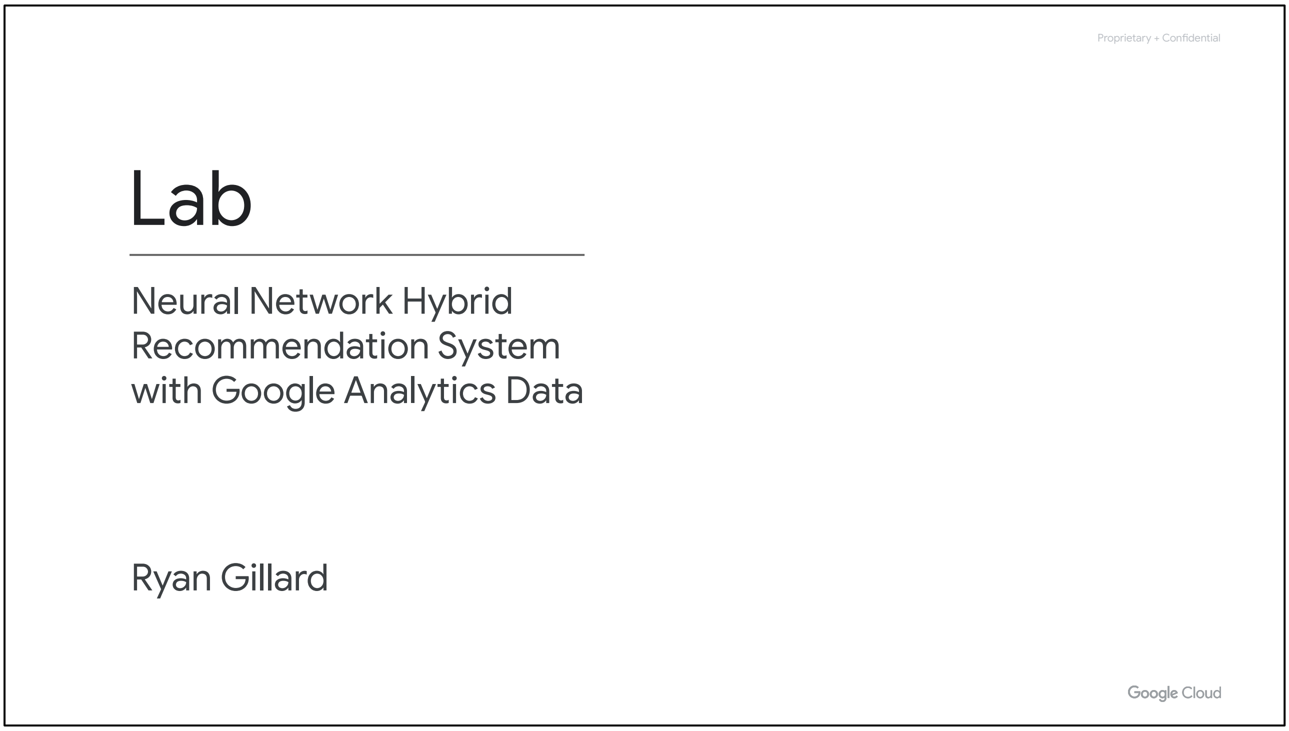


Quiz

What is important to have when making a hybrid recommendation system?

A. Data collection with recommendation in mind
B. Many different datasets
C. More than one recommendation model type
D. An ML model pipeline
[✓ E. All of the above]

The correct answer is E! Hybrid recommendation systems work best when they have a lot of data from multiple sources. This includes user-entered data, data entered by product experts for items, and user-item interaction data. Therefore, it is important when designing a website, service, etc., that you keep this all in mind, so that you will have the data available when creating your recommendation system. This is a lot better than making a recommendation system and not having the data ready to make it come to life. It's also important to have more than one recommendation model type. Each type comes with its own pros and cons, and by using a combination of them, you can eliminate or at least mitigate most of the cons. These multiple models being fed by multiple data sources should be arranged into a machine learning model pipeline that uses a deep neural network to combine them in the best way to give great recommendations.

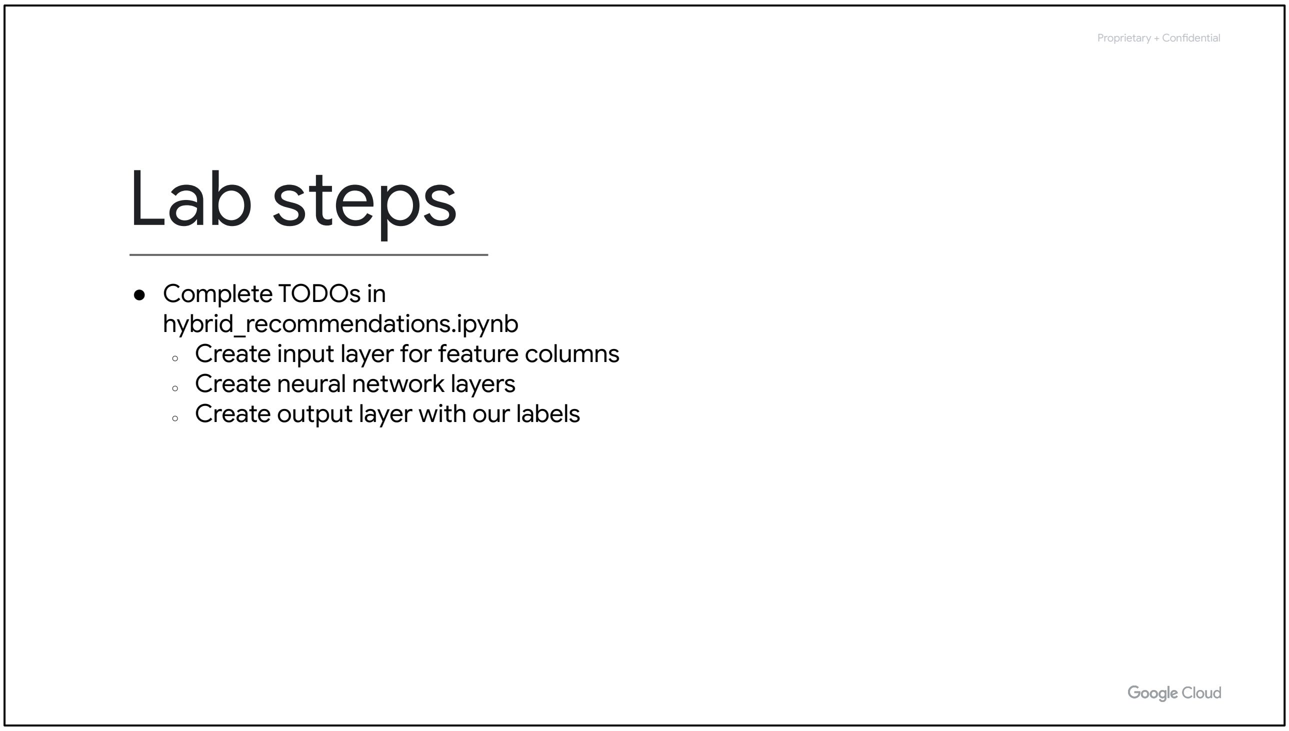


Lab

Neural Network Hybrid Recommendation System with Google Analytics Data

Ryan Gillard

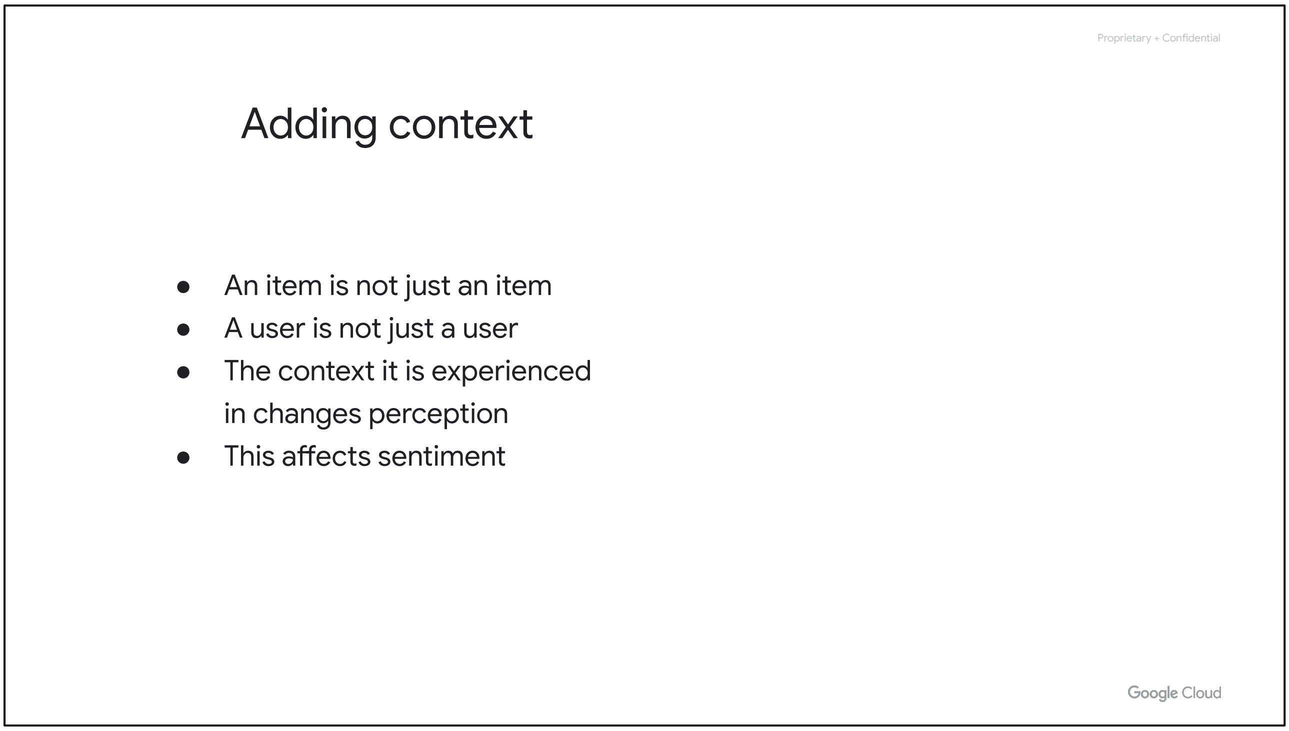
Now it's time to see whether we can combine all of this together. In this lab, we will be making a neural network hybrid recommendation system, using again Google Analytics data.



Lab steps

- Complete TODOs in hybrid_recommendations.ipynb
  - Create input layer for feature columns
  - Create neural network layers
  - Create output layer with our labels

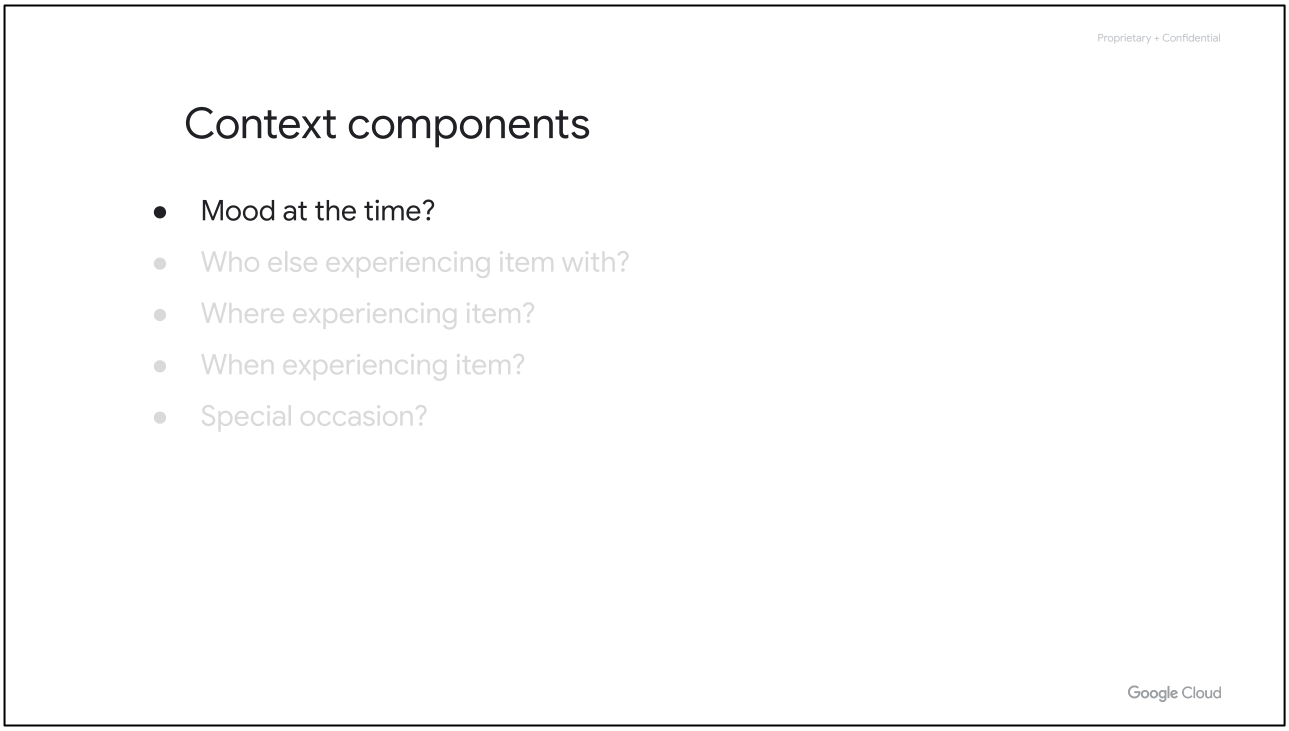
In this lab, we will be completing the TODOs in the hybrid_recommendations notebook. We will need to create an input layer for our feature columns, using our joined datasets. We will then need to connect this together with a deep neural network via layers. Lastly, we will need to create our output layer with our labels so that we can make our recommendations.



Adding context

- An item is not just an item
- A user is not just a user
- The context it is experienced in changes perception
- This affects sentiment

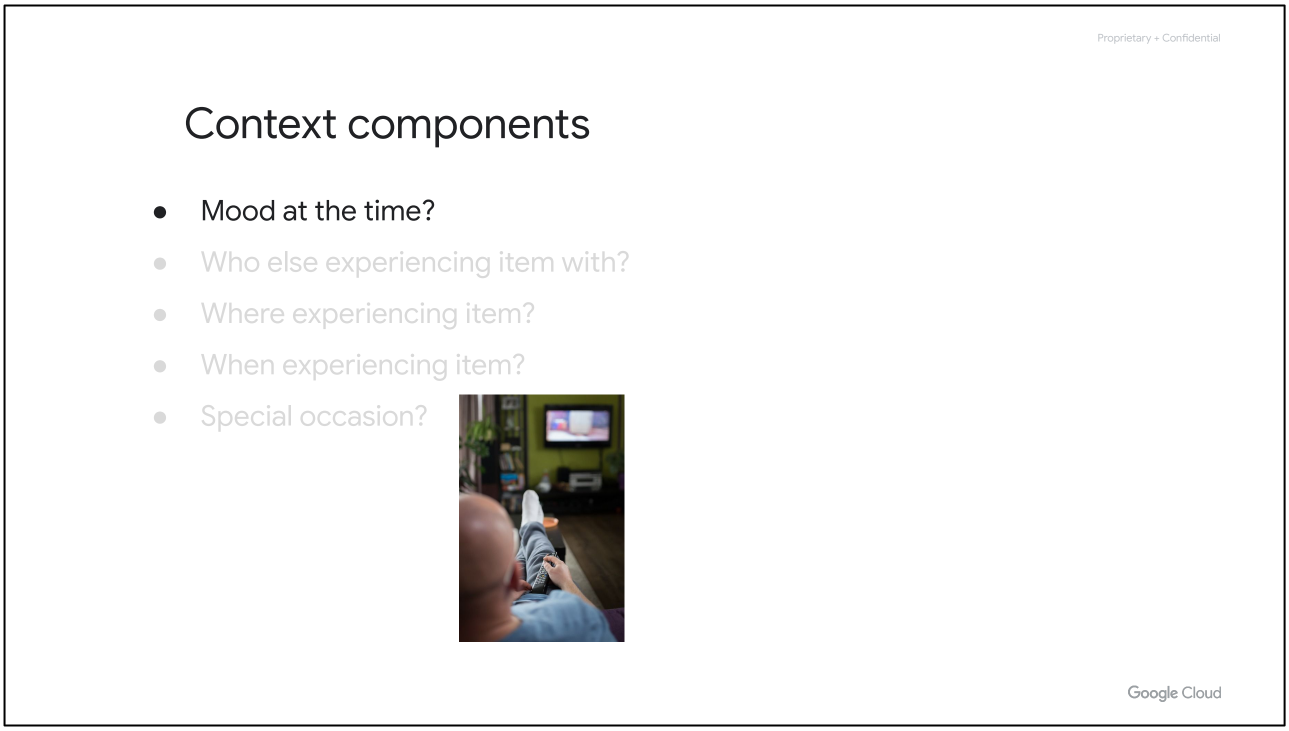
So we've seen how to combine the three main types of recommendation systems into a hybrid model, but some of these were somewhat stateless pieces of data. We don't know what was going on when someone was interacting with an item that could have changed how they felt about it. The context it was experienced in matters. If the context was different, then the perception of enjoyment could be different as well. This change in perception affects sentiment, which can change the feedback provided by the user about the item. Therefore, an item is not just an item, and a user is not just a user. There is more nuance to them, and that comes from context.



Context components

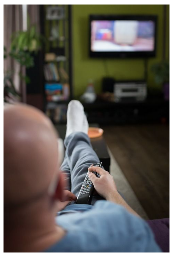
- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

Context is not simple. It is built up of many components that a user is remembering from the past, experiencing in the present, and looking forward to in the future. Imagine watching a movie. What is your mood at the time?



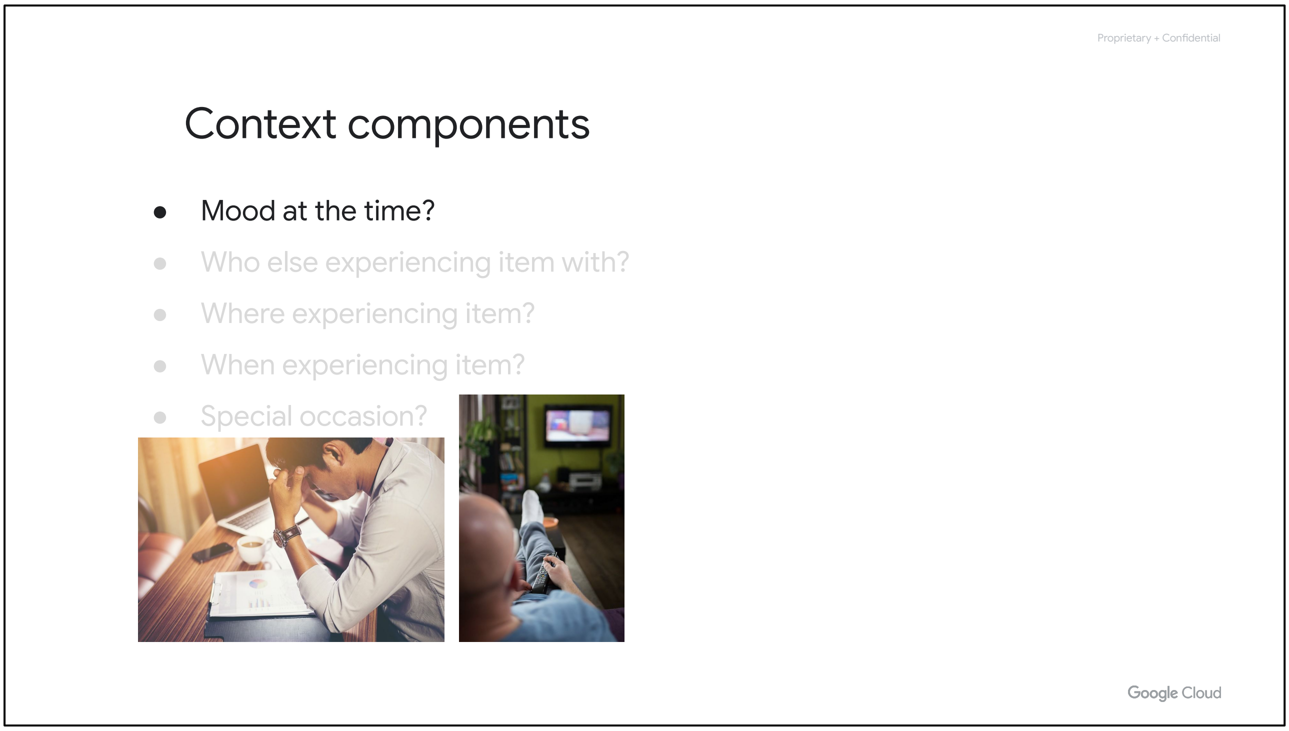
Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?



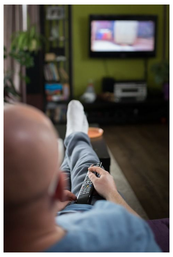
Are you super relaxed, curled up on the couch with a bowl of ice cream?

https://pixabay.com/en/television-and-radio-relaxation-2741799/ cc0



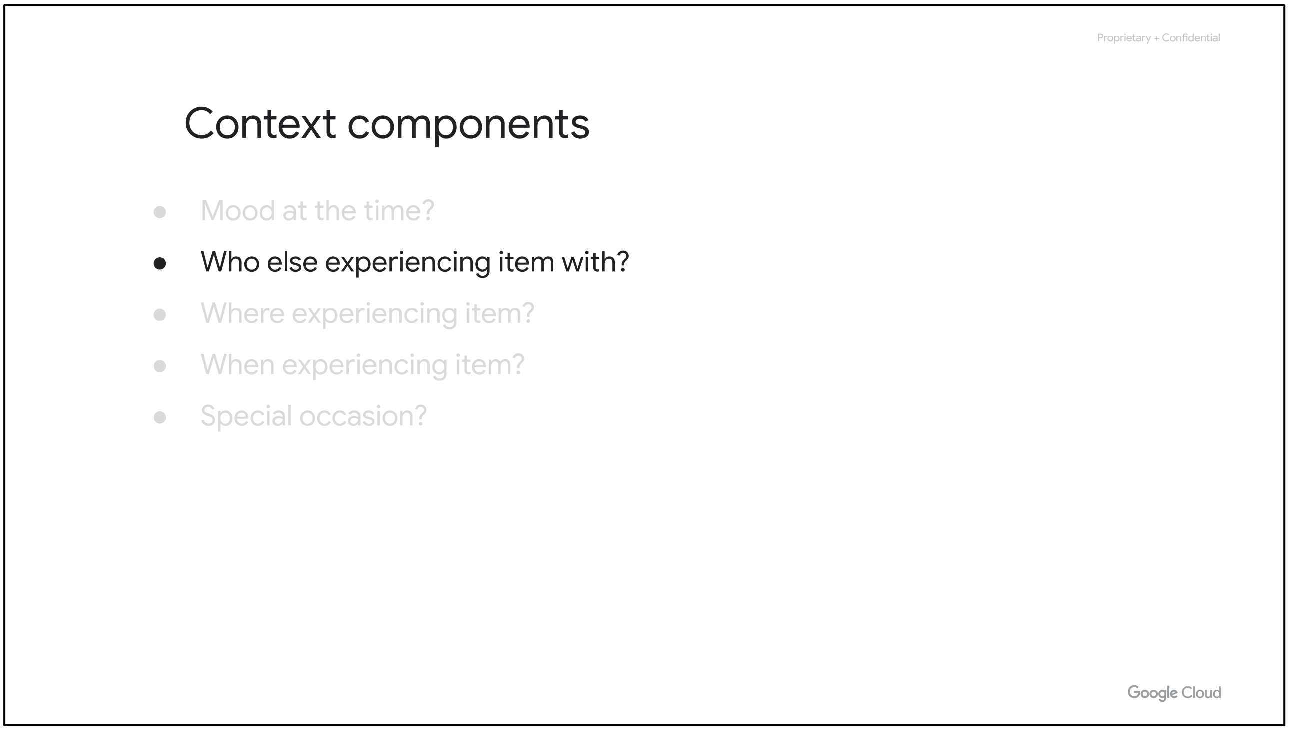
Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?



Are you completely tired from a long day at work, stressed from a long commute home, and worried about finishing your project by the deadline tomorrow? This probably makes a difference in how you perceive and enjoy the movie.

https://pixabay.com/en/television-and-radio-relaxation-2741799/ cc0
https://pixabay.com/en/work-stressed-accounts-man-working-2005640/ cc0



Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

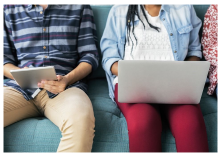
Google Cloud

It also depends on who you are experiencing the item with. Back to our movie example.

https://pixabay.com/en/boy-break-browsing-casual-computer-1986107/ cc0

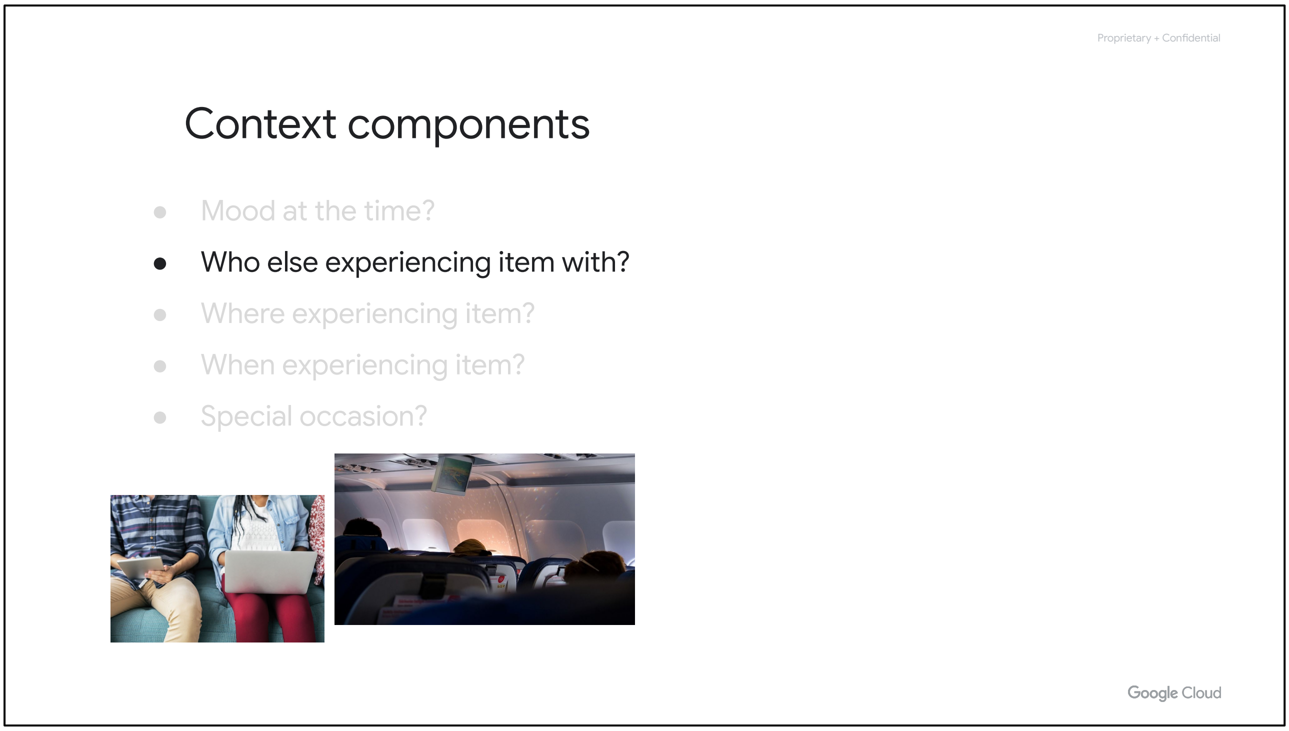
Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?



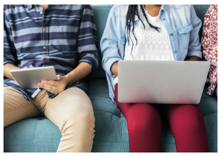
What if you are watching a movie with your kids on the couch? What if instead it is on the couch with your partner?

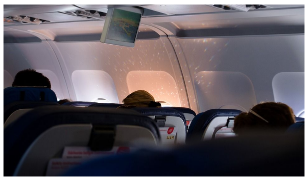
https://pixabay.com/en/boy-break-browsing-casual-computer-1986107/ cc0



Context components

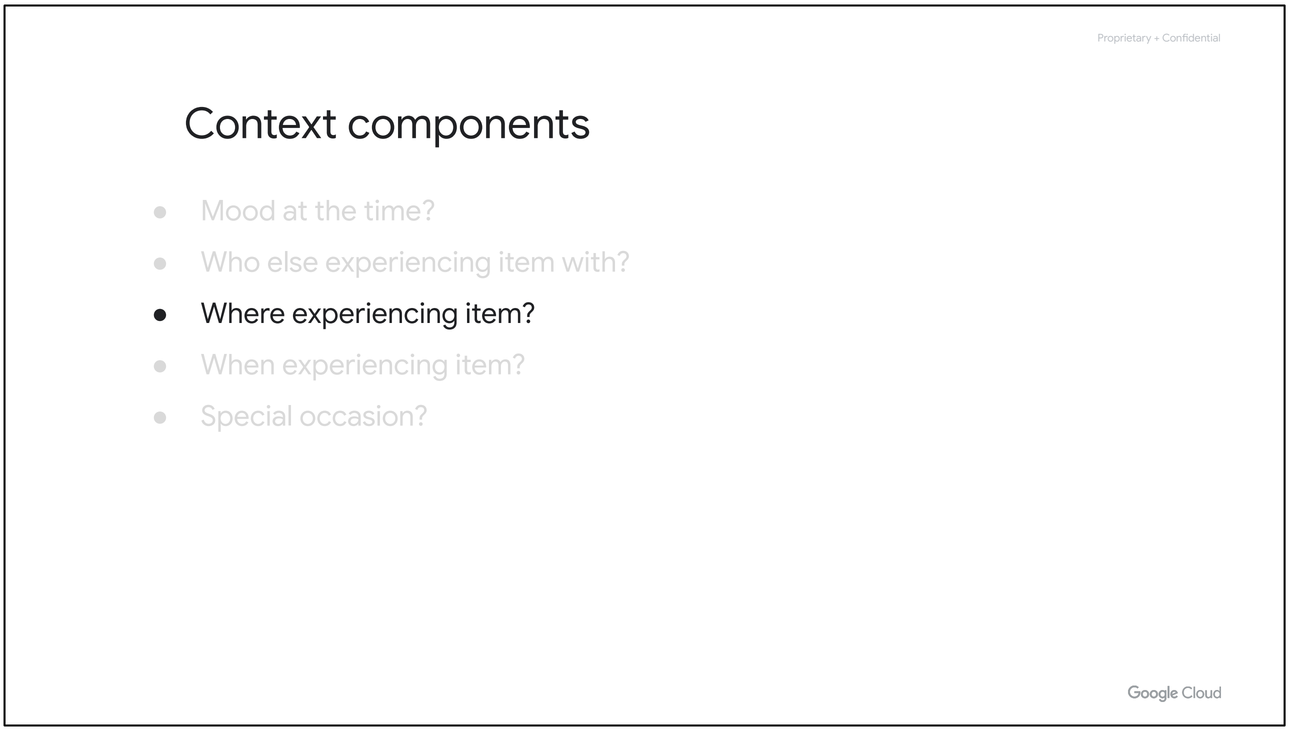
- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?





What if instead, it is crammed into an economy seat on an airplane with a bunch of random strangers? The experience will be totally different depending who you are with, and can affect how you would rate the item.

https://pixabay.com/en/boy-break-browsing-casual-computer-1986107/ cc0
https://pixabay.com/en/inside-airline-airplane-seats-2568698/ cc0


Proprietary + Confidential

Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

Google Cloud

Location, location, location. Where we experience things matters a lot when it comes to our perception.

Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

Watching a movie at home is a different experience from watching it at the movie theater.

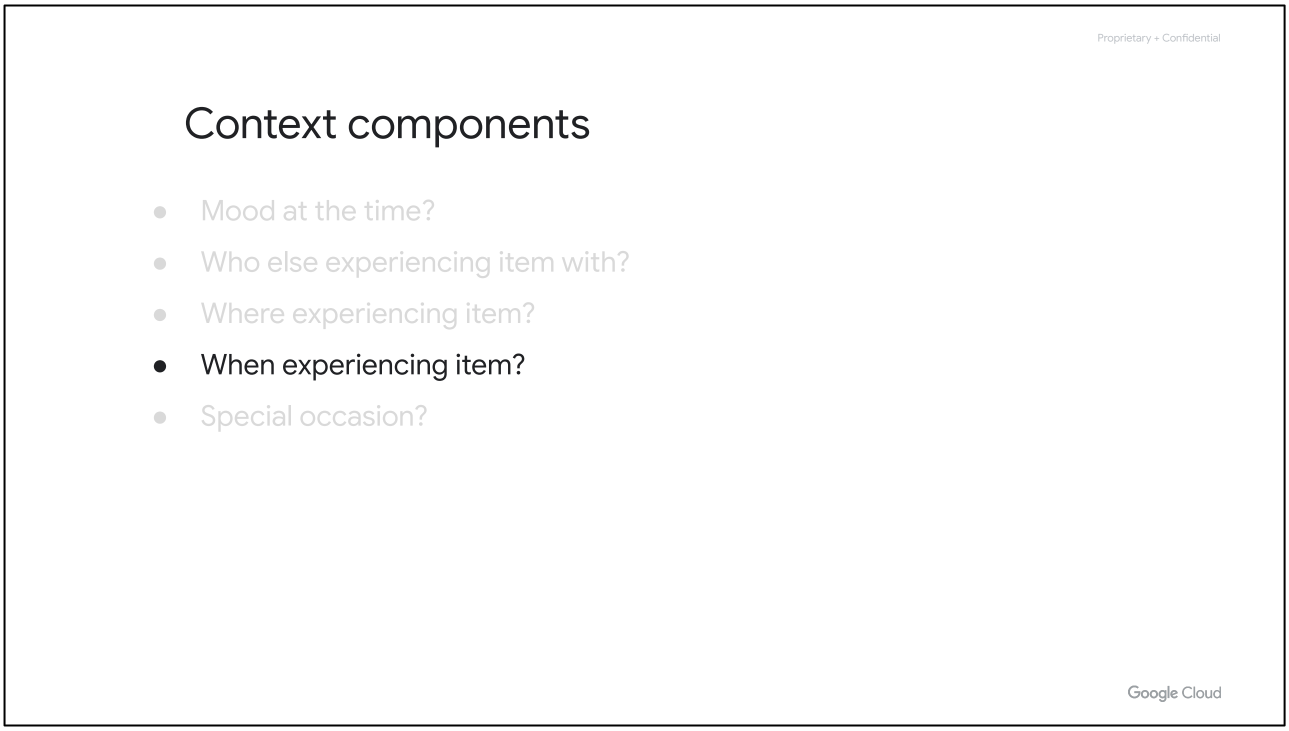
https://pixabay.com/en/children-tv-child-television-home-403582/ cc0

Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

Watching a movie on a pop-up big screen under the stars camping in the desert is totally different and will affect our rating as well.

https://pixabay.com/en/children-tv-child-television-home-403582/ cc0
https://pixabay.com/en/milky-way-stars-night-sky-923738/ cc0



Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

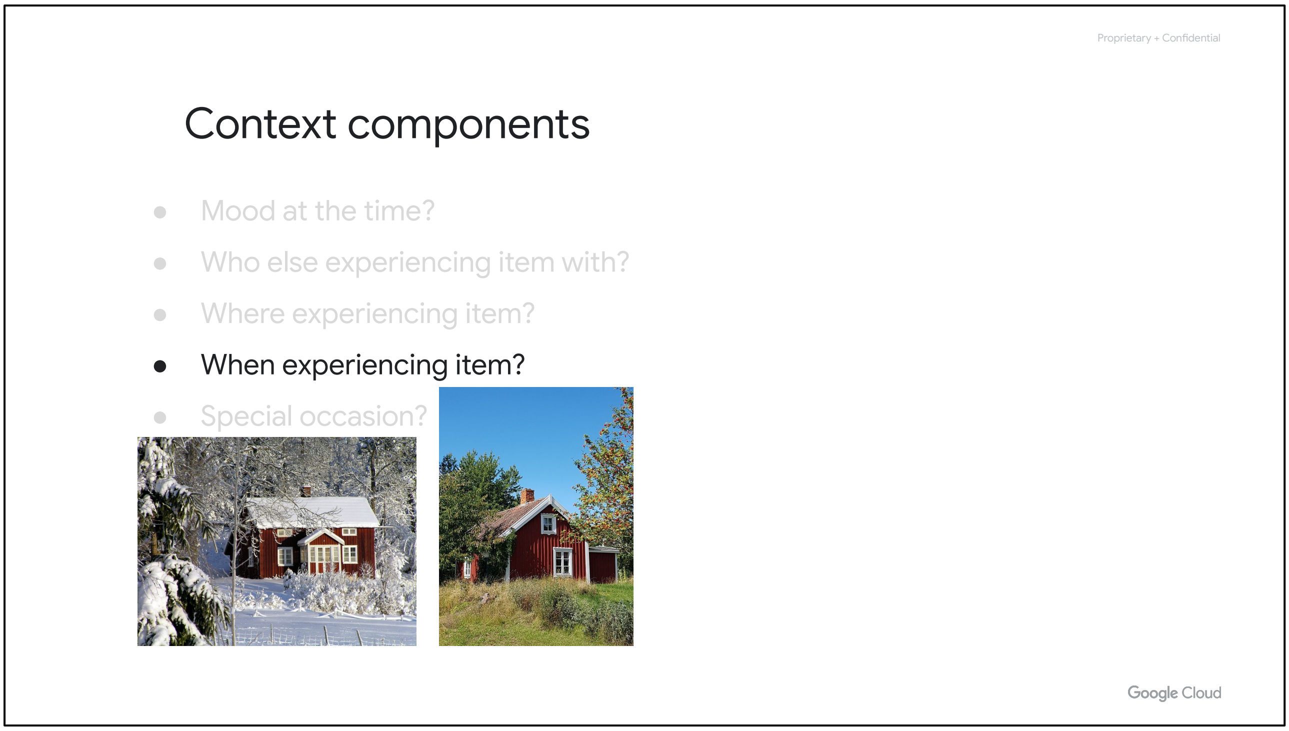
Time is also a factor in how we judge our experiences. Imagine we can travel to the same place but we can go in winter or summer. These can be drastically different experiences.

Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

Let's say we stay at a bed and breakfast in the hot summer and the air conditioning is broken the whole time, so we give them a low rating.

https://pixabay.com/en/torp-summer-cottage-cottage-2672839/ cc0

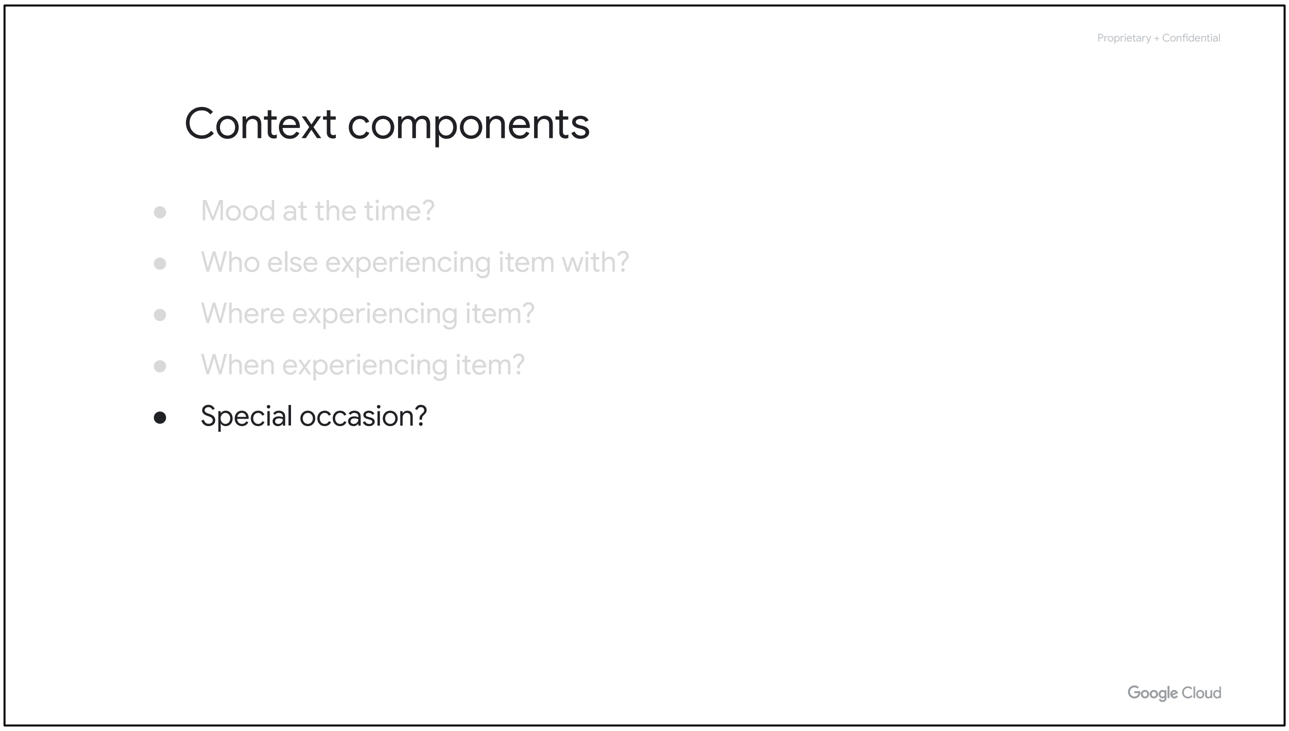


Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

However, if we chose to go in the cold winter we give it a high rating, because they always provided a delicious hot breakfast and the house was kept nice and cozy with a great furnace. Even if the air conditioner had been broken in our winter example, that component of the bed and breakfast we didn't interact with, because it wasn't needed, and therefore, didn't influence our rating. Same place, different times, different ratings.

https://pixabay.com/en/torp-summer-cottage-cottage-2672839/ cc0
https://pixabay.com/en/house-cottage-winter-red-cottage-2151102/ cc0



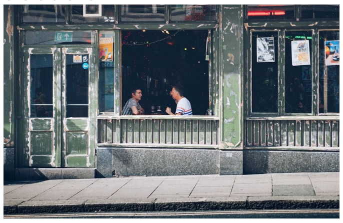
Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

Perhaps it is a special occasion? Special occasions can heighten our sensitivity to experience because there are emotions, expectations, etc.

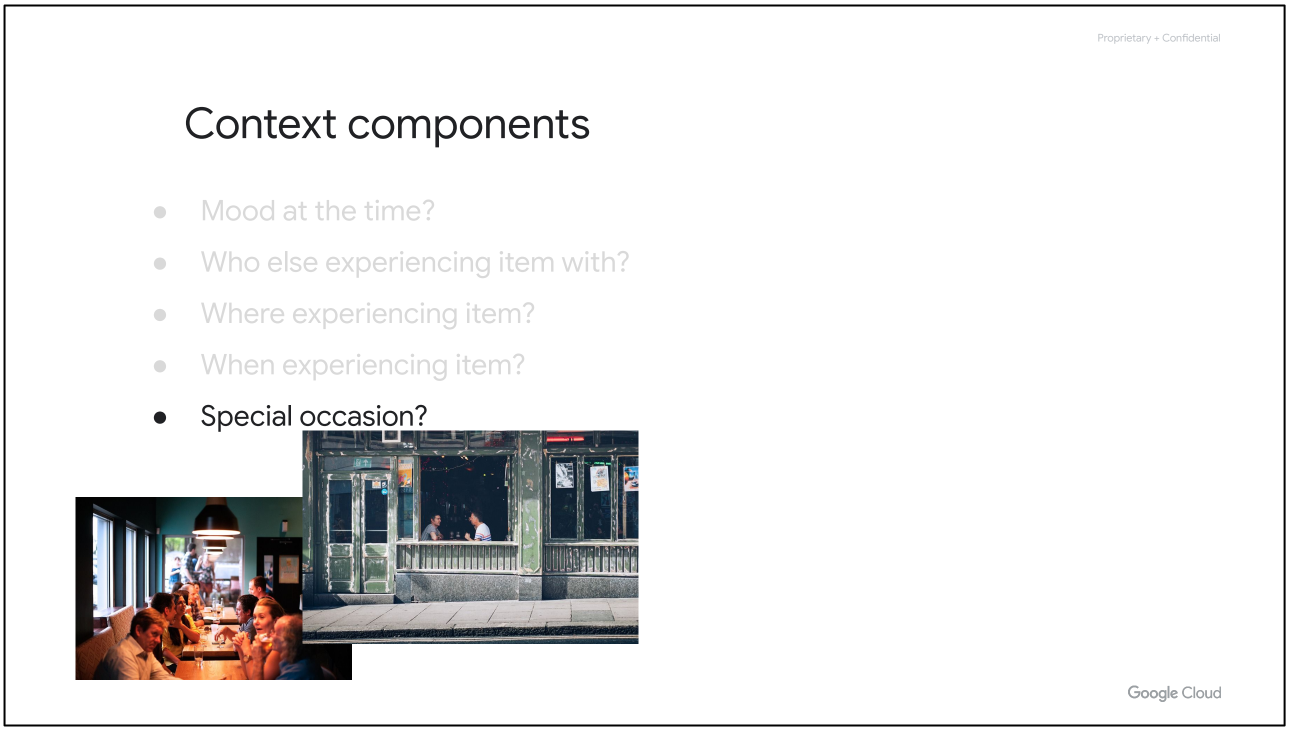
Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?



Going to a restaurant for a quick lunch with a friend is highly different...

https://pixabay.com/en/people-men-friends-breakfast-lunch-2606147/ cc0

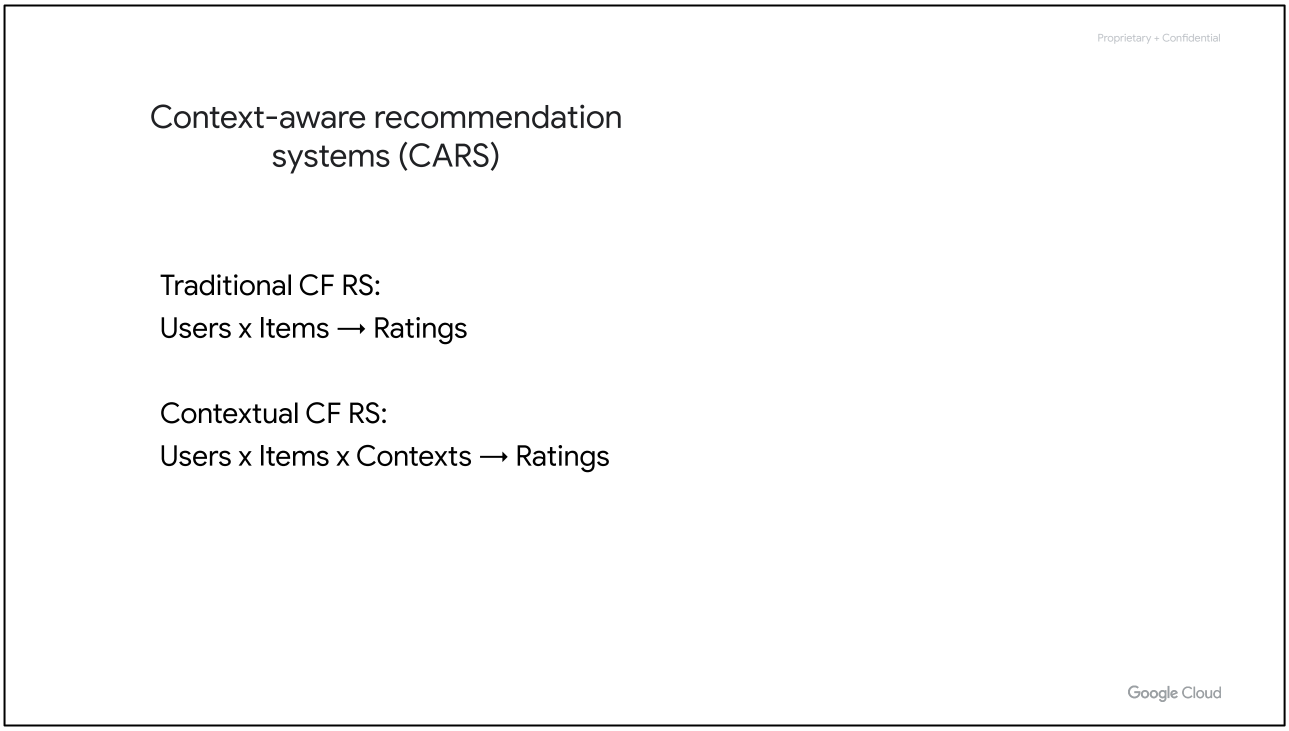


Context components

- Mood at the time?
- Who else experiencing item with?
- Where experiencing item?
- When experiencing item?
- Special occasion?

...from going to the same place for your wedding rehearsal dinner with your entire family the night before the big day. Even going to a restaurant on your birthday is a special event, hence why a lot of restaurants offer free desserts, come sing a birthday song, etc., because they want you to have a good experience, especially because your special occasion changes your perception and you may be more likely to rate more strongly one way or the other.

https://pixabay.com/en/people-men-friends-breakfast-lunch-2606147/ cc0
https://pixabay.com/en/restaurant-people-eating-690975/ cc0



Context-aware recommendation systems (CARS)

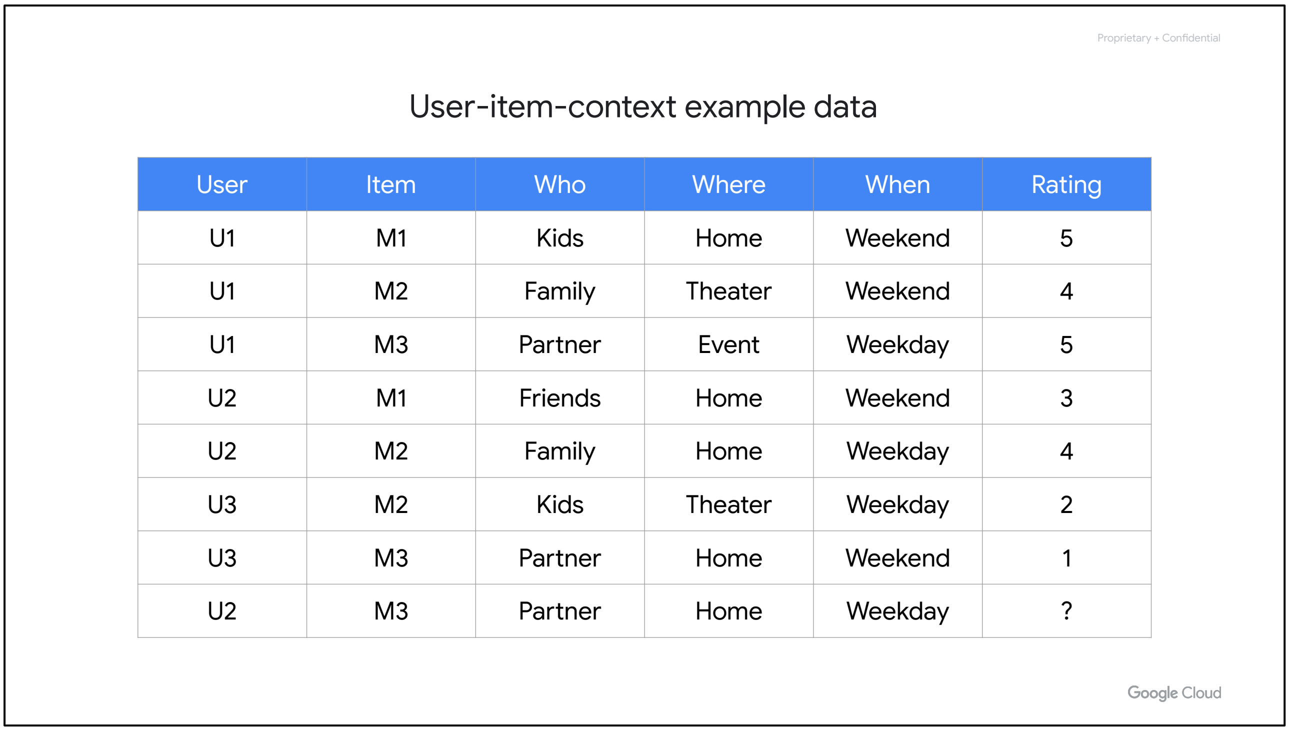
Traditional CF RS:
Users x Items ⟶ Ratings

Contextual CF RS:
Users x Items x Contexts ⟶ Ratings

Context-aware recommendation systems, or CARS, add an extra dimension to our usual collaborative filtering problem.

Traditional collaborative filtering recommendation systems use a rank 2 tensor, a user-item interaction matrix, containing explicit or implicit ratings.

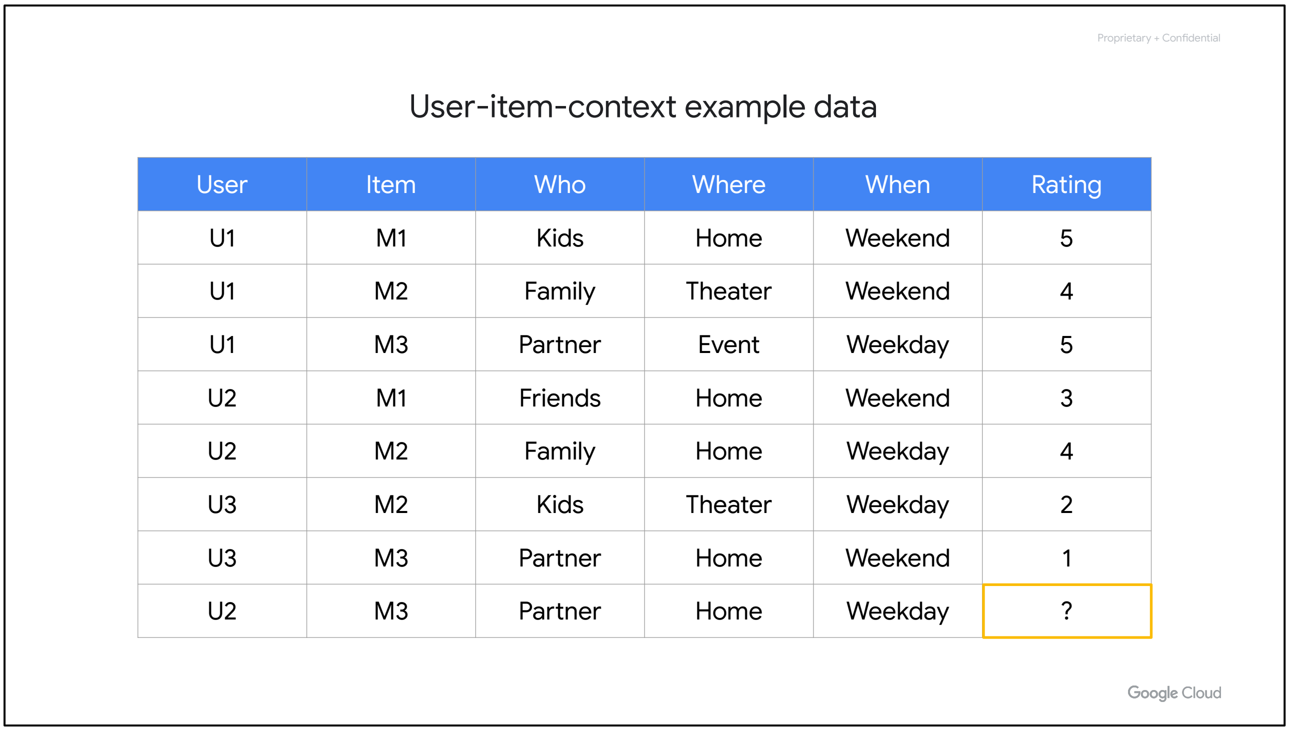
Contextual collaborative filtering recommendation systems, on the other hand, use a rank 3 plus tensor, where the user-item interaction matrix, full of ratings, is stratified across multiple dimensions of context.



User-item-context example data

User	Item	Who	Where	When	Rating
U1	M1	Kids	Home	Weekend	5
U1	M2	Family	Theater	Weekend	4
U1	M3	Partner	Event	Weekday	5
U2	M1	Friends	Home	Weekend	3
U2	M2	Family	Home	Weekday	4
U3	M2	Kids	Theater	Weekday	2
U3	M3	Partner	Home	Weekend	1
U2	M3	Partner	Home	Weekday	?

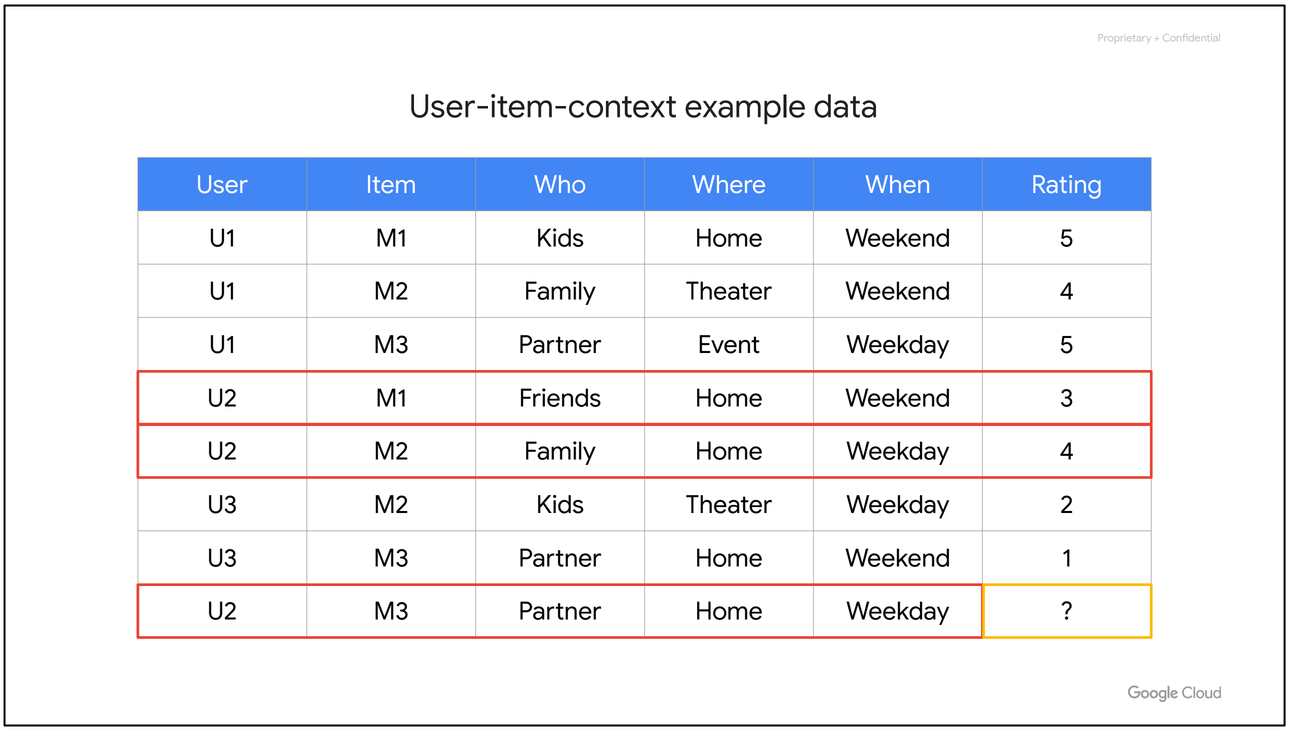
Here's an example of some user-item-context data for movies. There is a user column that lists the userIds, an item column that lists the movieIds, a who column that lists who the user watched the movie with, a where column that lists where the user watched the movie, a when column that lists whether the movie was watched on a weekday or weekend, and lastly a rating column that lists the explicit rating left for that user. As you can see, this data is much more complex than just the typical user, item, rating matrix we are used to.



User-item-context example data

User	Item	Who	Where	When	Rating
U1	M1	Kids	Home	Weekend	5
U1	M2	Family	Theater	Weekend	4
U1	M3	Partner	Event	Weekday	5
U2	M1	Friends	Home	Weekend	3
U2	M2	Family	Home	Weekday	4
U3	M2	Kids	Theater	Weekday	2
U3	M3	Partner	Home	Weekend	1
U2	M3	Partner	Home	Weekday	?

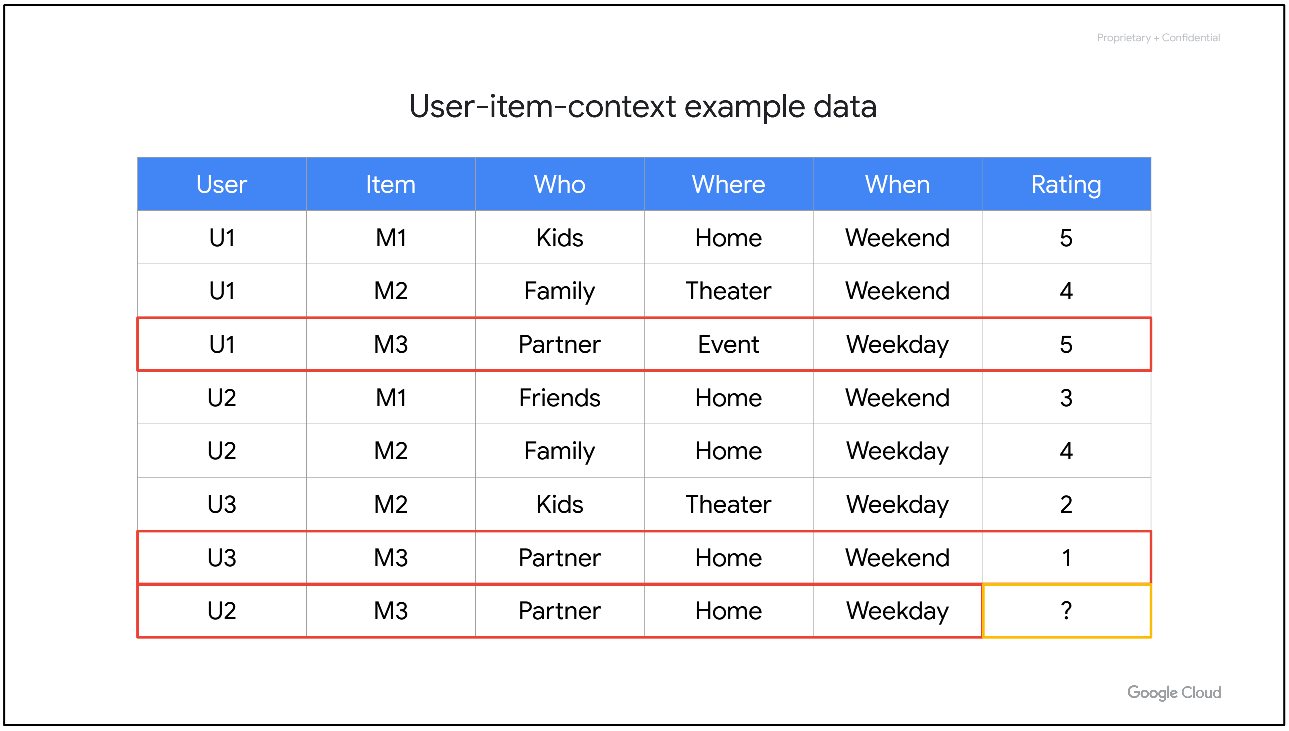
We might notice that this table is not complete. We want to predict what user 2 will rate movie 3 if they see the movie with their partner, at home, on a weekday. With the traditional method, we would just want to predict what user 2 will rate the item altogether, not considering context. But hopefully context here will allow us to better predict how they would rate the movie within that situation. Note, this predicted rating could be different from a rating predicted for the same user and item, but with a different context. For example, it could be a different rating for user 2 for movie 3, with their partner, at home, BUT now on a weekend. Plus all of the other permutations. So how does this work?



User-item-context example data

User	Item	Who	Where	When	Rating
U1	M1	Kids	Home	Weekend	5
U1	M2	Family	Theater	Weekend	4
U1	M3	Partner	Event	Weekday	5
U2	M1	Friends	Home	Weekend	3
U2	M2	Family	Home	Weekday	4
U3	M2	Kids	Theater	Weekday	2
U3	M3	Partner	Home	Weekend	1
U2	M3	Partner	Home	Weekday	?

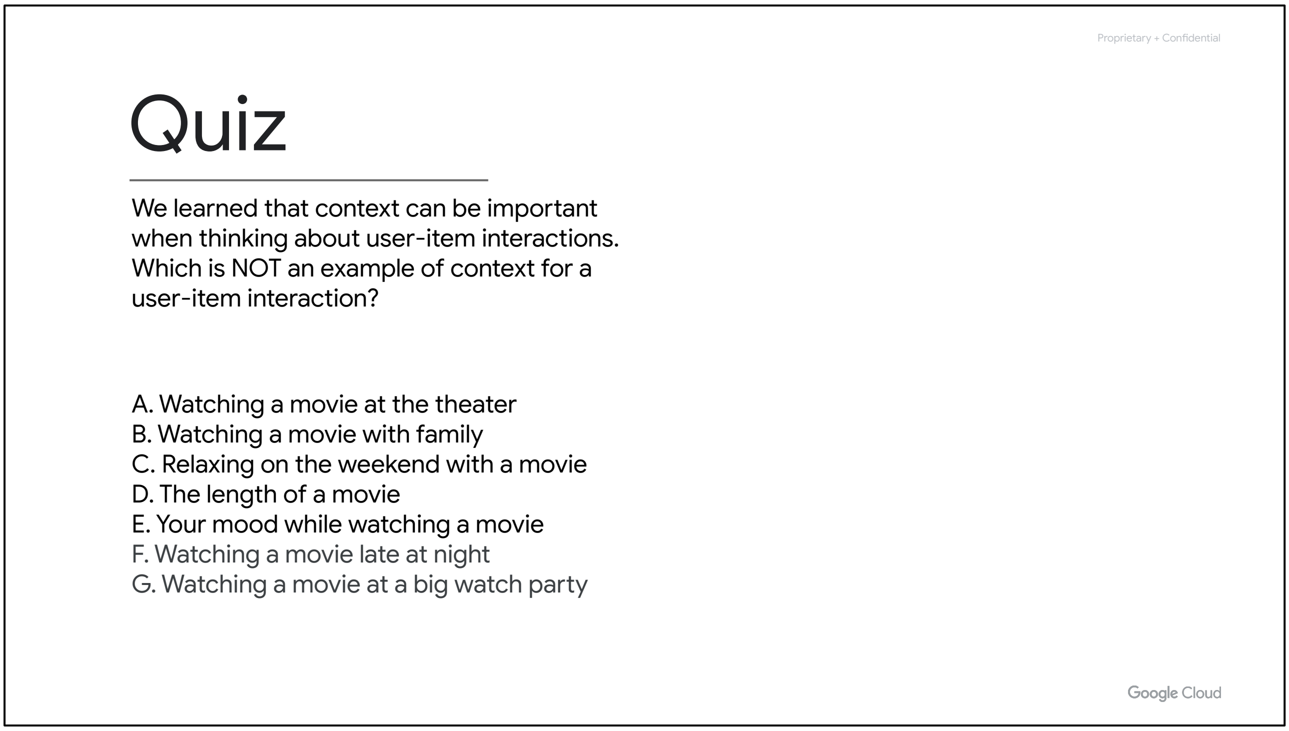
We should first look at the user of interest's interaction history. As you can see here, user 2 has also watched movies 1 and 2. Movie 1 was with friends at home on the weekend, and they gave a rating of 3. Movie 2 was with family at home on a weekday, and they gave a rating of 4. None of the contexts are exactly the same as our movie of interest, but both were seen at home, which matches our target location, and movie 2 was seen on a weekday, just like our rating of interest.



User-item-context example data

User	Item	Who	Where	When	Rating
U1	M1	Kids	Home	Weekend	5
U1	M2	Family	Theater	Weekend	4
U1	M3	Partner	Event	Weekday	5
U2	M1	Friends	Home	Weekend	3
U2	M2	Family	Home	Weekday	4
U3	M2	Kids	Theater	Weekday	2
U3	M3	Partner	Home	Weekend	1
U2	M3	Partner	Home	Weekday	?

We also should look at the item of interest's user interaction history. We can see that both user 1 and user 3 have rated this movie. User 1 watched it with their partner at an event on a weekday and gave it a rating of 5. User 3 watched the movie also with their partner, but now, at home and now on the weekend, and only gave it a rating of 1. Once again, none of the contexts perfectly match to our target rating, but individually all three are with their partners, user 2 also saw the movie at home, and user 1 also saw the movie on a weekday. So how do we algorithmically use this data to predict ratings for movie recommendation?

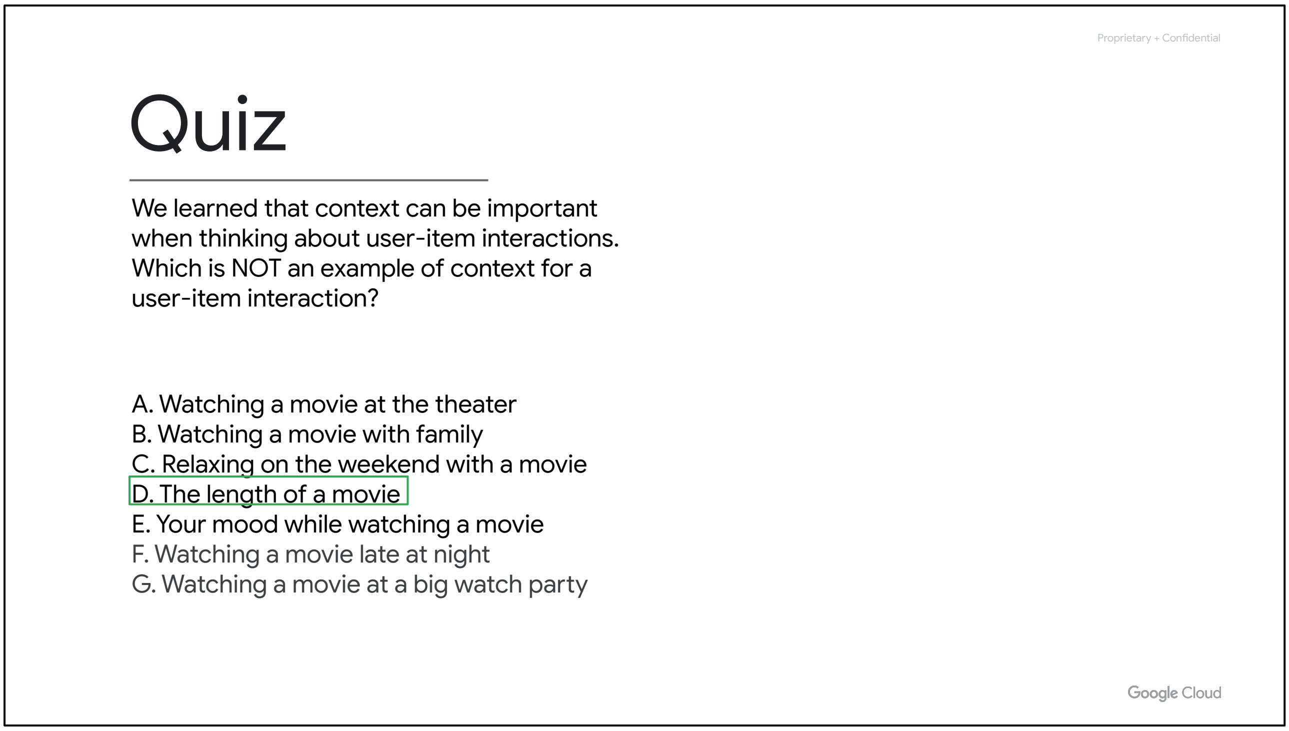


Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
D. The length of a movie
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Now that we've learned about how context can affect user-item interactions, let's test our knowledge! We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction? Watching a movie at the theater? Watching a movie with family? Relaxing on the weekend with a movie? The length of a movie? Your mood while watching a movie? Watching a movie late at night? Or watching a movie at a big watch party?



Proprietary + Confidential

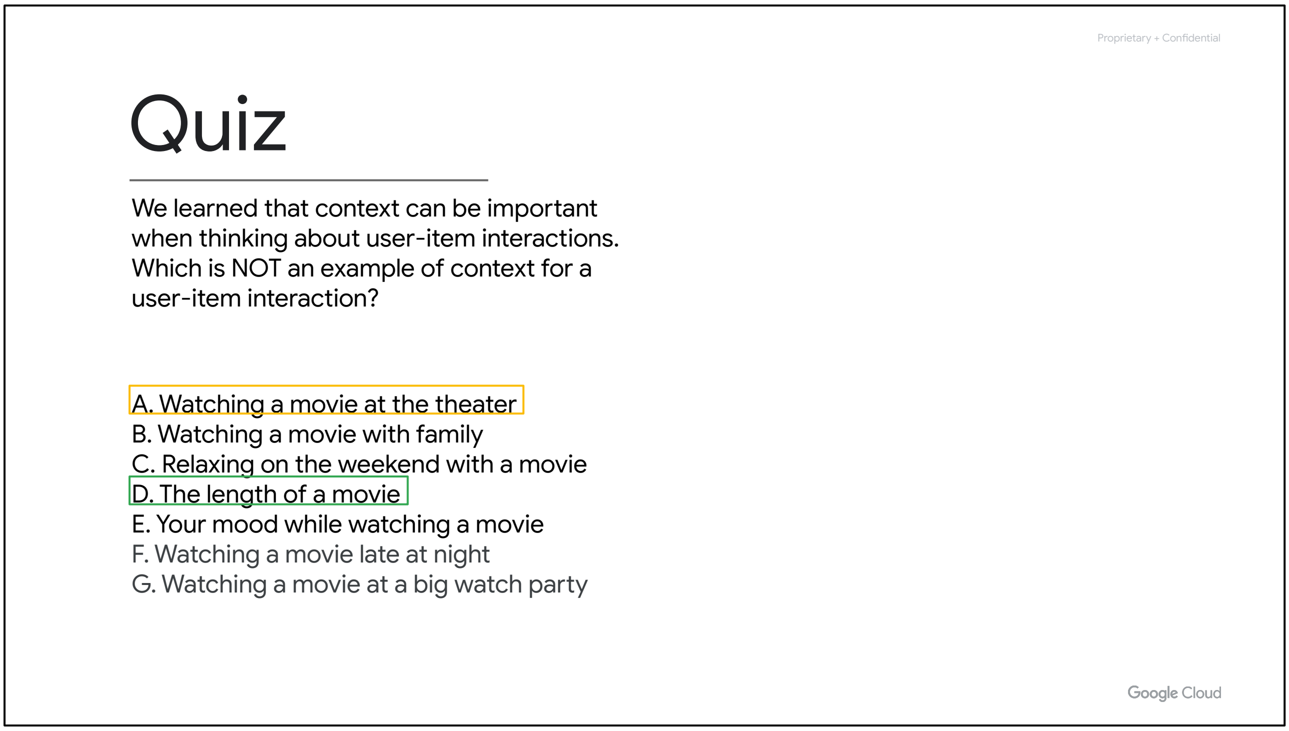
Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
[✓ D. The length of a movie]
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Google Cloud

The correct answer is D! The length of a movie is a property of the item itself, not the context it is experienced in. That is not to say that context can't have a role related to the length of a movie though. Perhaps, the user is relaxing on a weekend and they enjoy a very long movie. However, if they were to have watched that same movie after a stressful day at work, when they need to start getting dinner ready for the family, they might get frustrated at its length, because they just want to get to the ending already. The length itself isn't context, it is a component of the item that is evaluated differently based on context.


Proprietary + Confidential

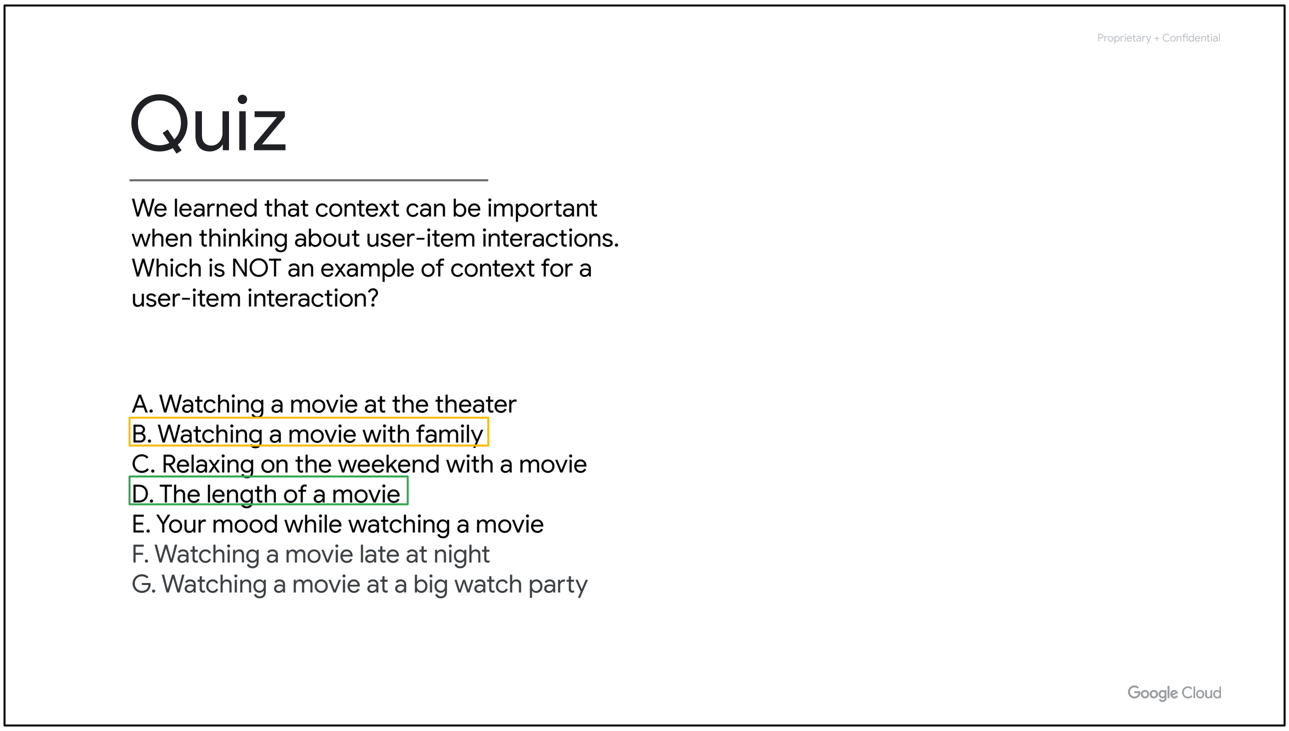
Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
D. The length of a movie
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Google Cloud

Watching a movie at the theater is a great example of location changing the context of your experience. Of course, the other factors still come in to play like who is also watching at the theater, when the user went to the theater, was it a special screening, etc.?

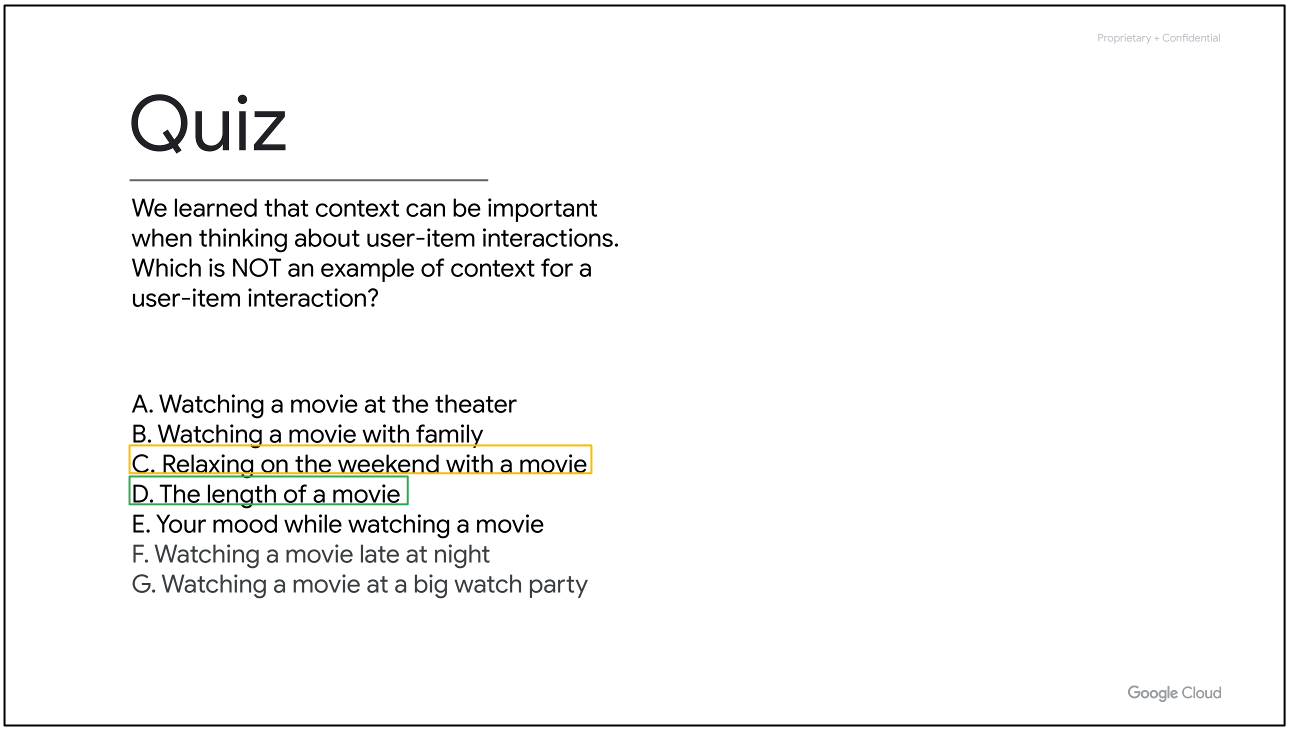


Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
[✓ D. The length of a movie]
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Watching a movie with family is mostly hitting on the who part of context, but of course, where's it being watched at, when, etc. comes into play.

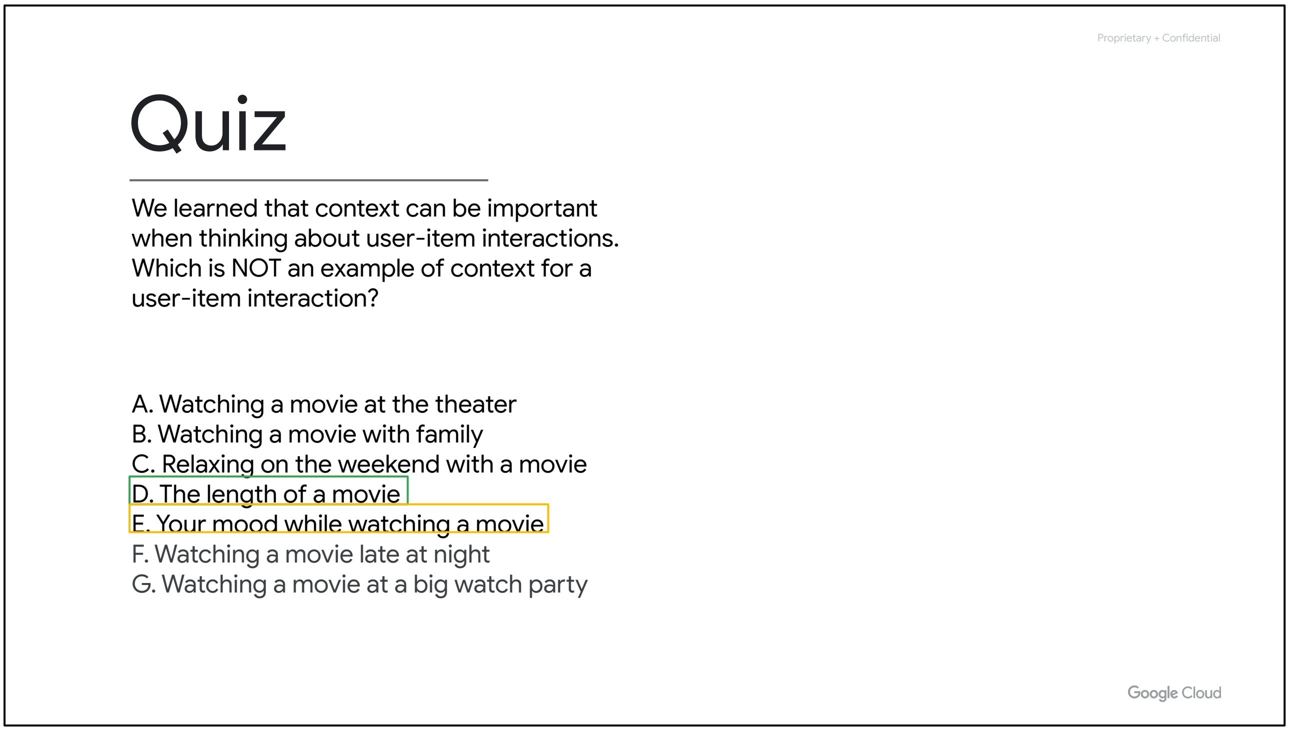


Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
D. The length of a movie
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Relaxing on the weekend with a movie taps mainly on the when aspect of context, but relaxing also hints that mood might be part of the context, not to mention where is the user relaxing, when, etc?

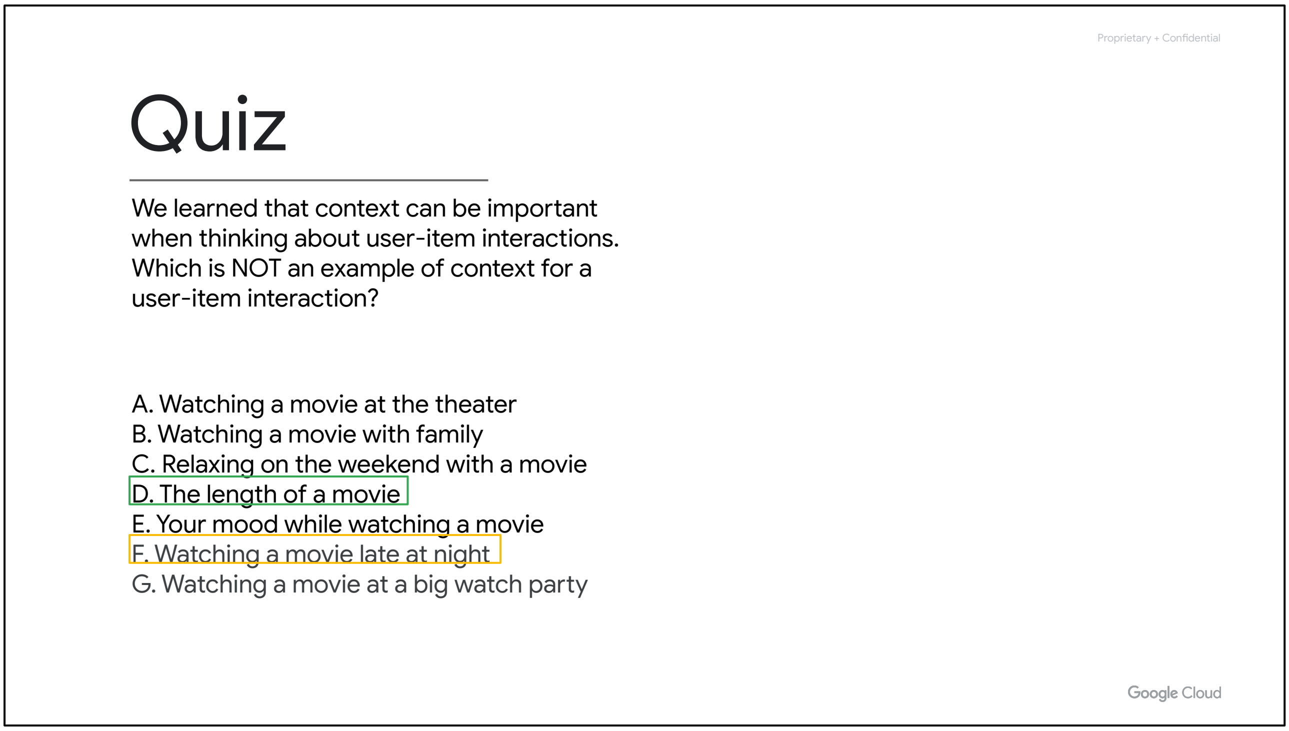


Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
D. The length of a movie
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Your mood while watching a movie definitely changes the context it is viewed in, and leads to differences in perception, as well as all of the other pieces of context going on during the movie.

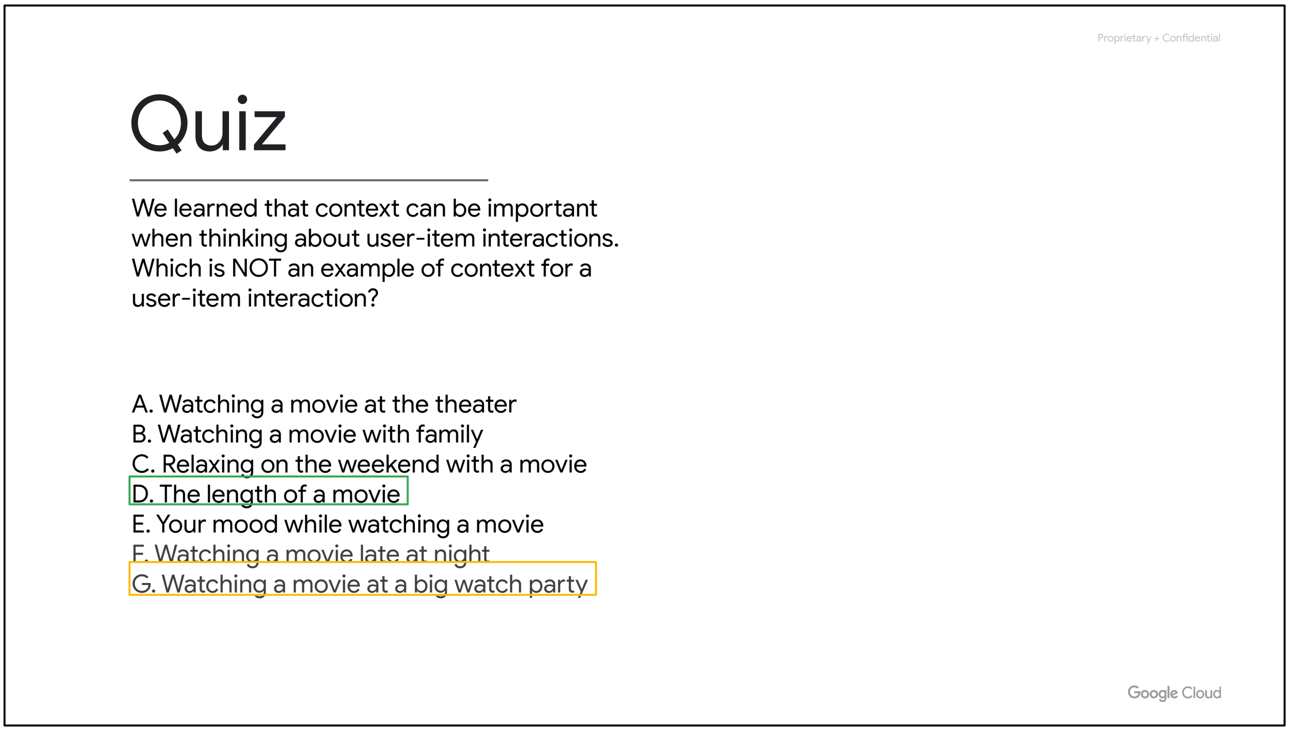


Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
[✓ D. The length of a movie]
E. Your mood while watching a movie
[✓ F. Watching a movie late at night]
G. Watching a movie at a big watch party

Watching a movie late at night once again mainly focuses on the context of time, but where is it being watched, with who, etc. always can affect the rating.



Proprietary + Confidential

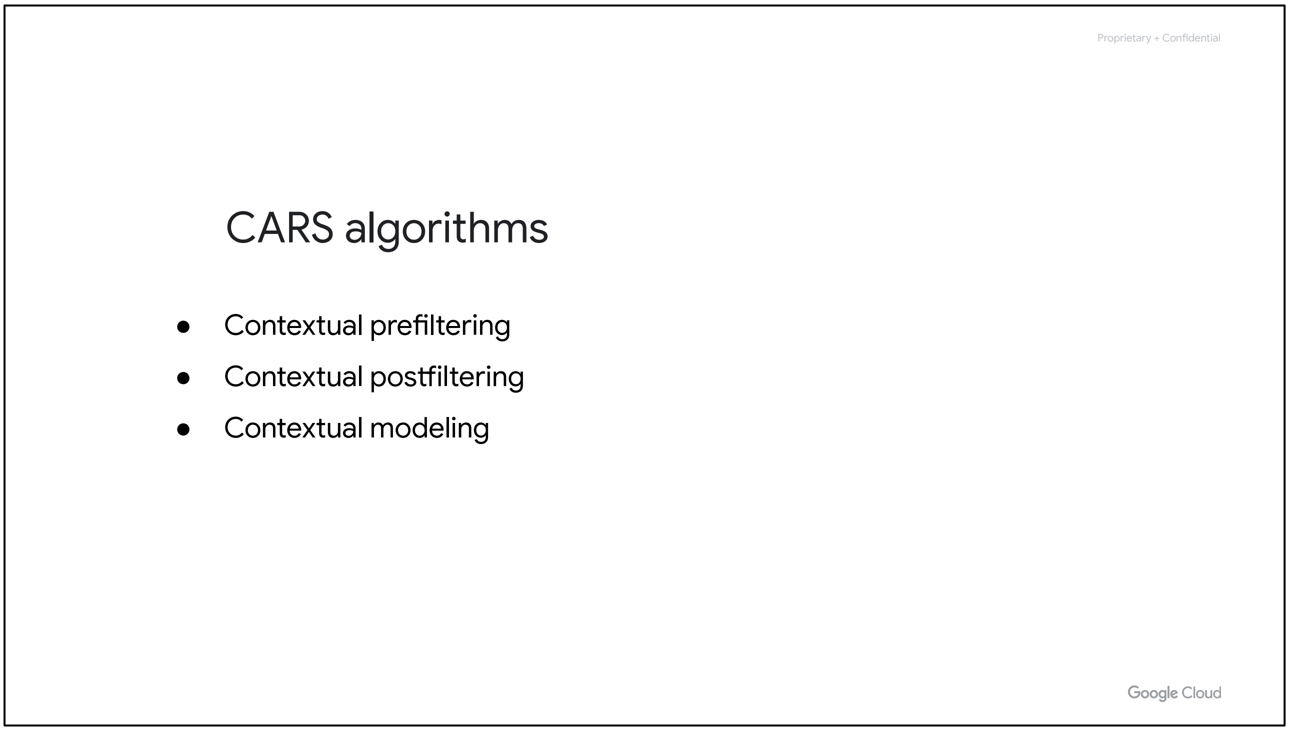
Quiz

We learned that context can be important when thinking about user-item interactions. Which is NOT an example of context for a user-item interaction?

A. Watching a movie at the theater
B. Watching a movie with family
C. Relaxing on the weekend with a movie
D. The length of a movie
E. Your mood while watching a movie
F. Watching a movie late at night
G. Watching a movie at a big watch party

Google Cloud

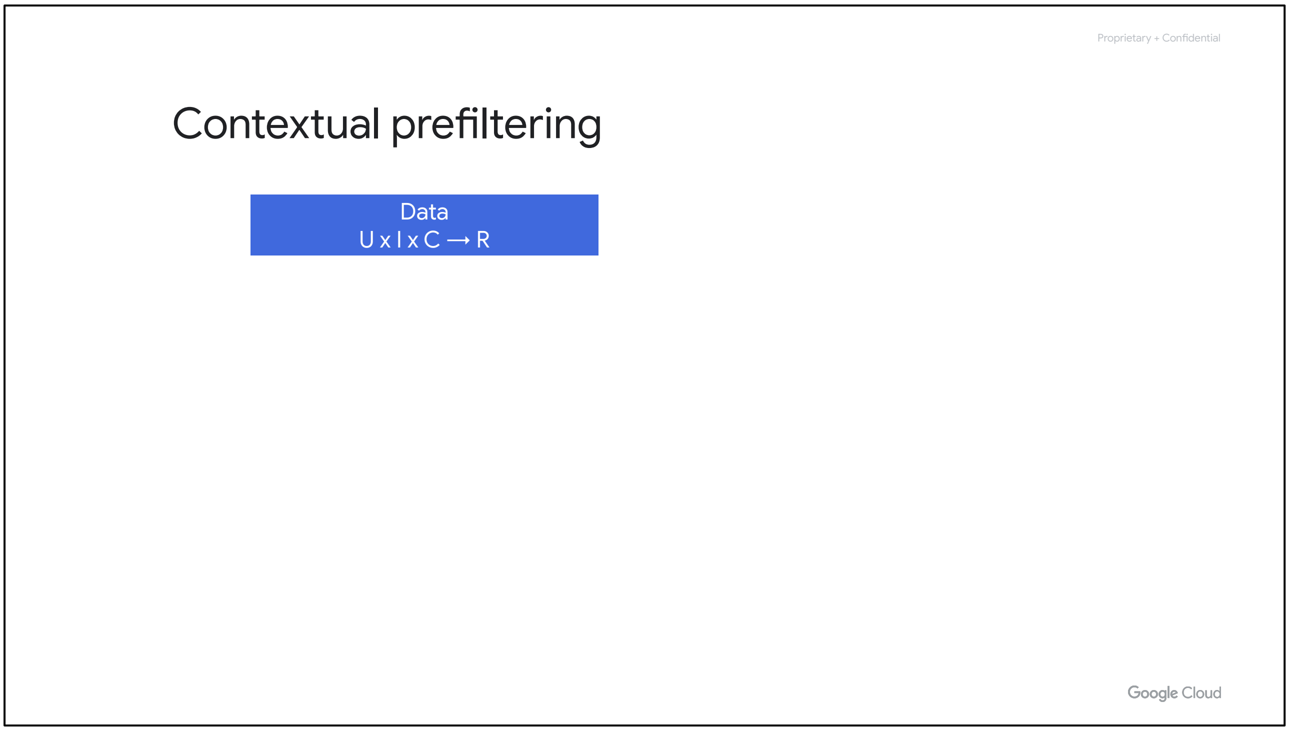
Lastly, watching a movie at a big watch party may be a special occasion and is a very different context than if the movie was watched with more typical conditions. As you can see, there never really is just one facet of context going on at once. All of them are happening simultaneously, and all can affect how a user perceives an item.



CARS algorithms

- Contextual prefiltering
- Contextual postfiltering
- Contextual modeling

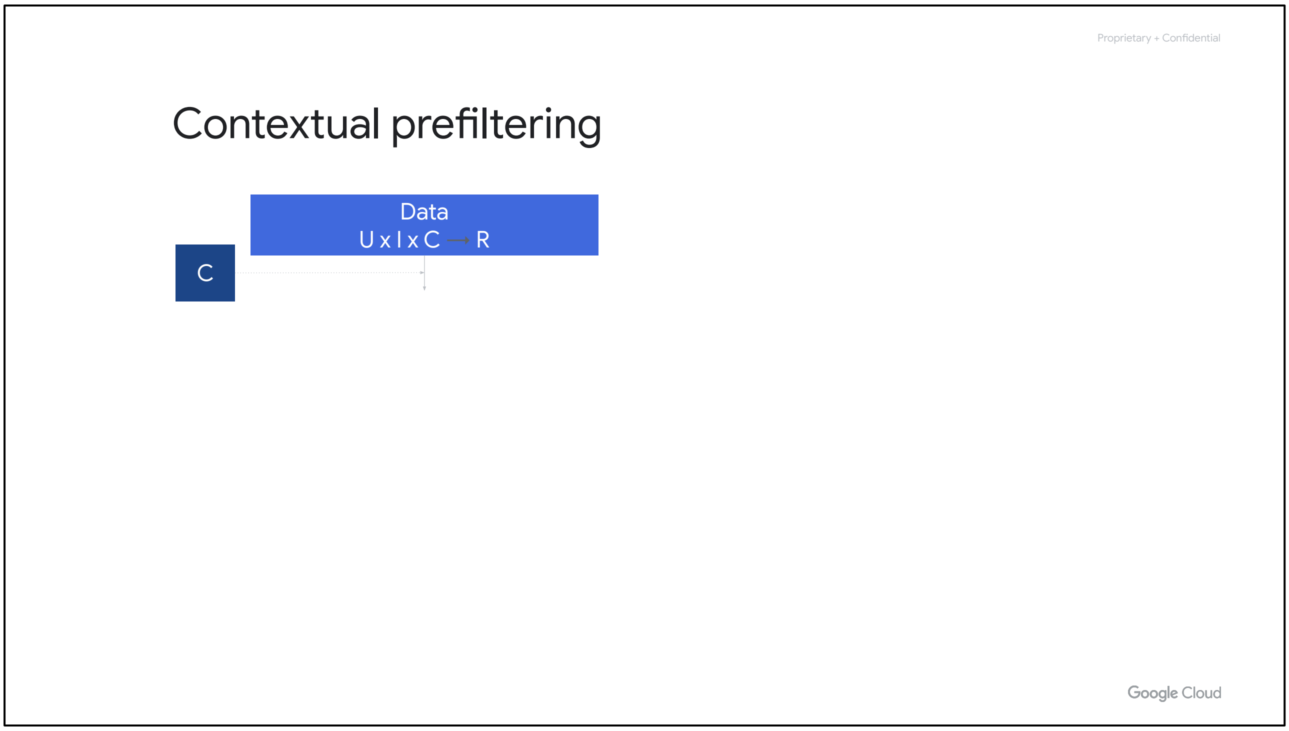
There are three main types of context-aware recommendation systems, or CARS, algorithms. There is contextual prefiltering, contextual postfitering, and contextual modeling. Let's go into more detail for each type.



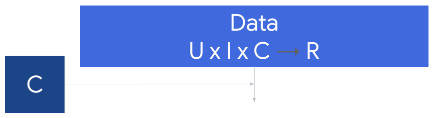
Contextual prefiltering

Data
U x I x C ⟶ R

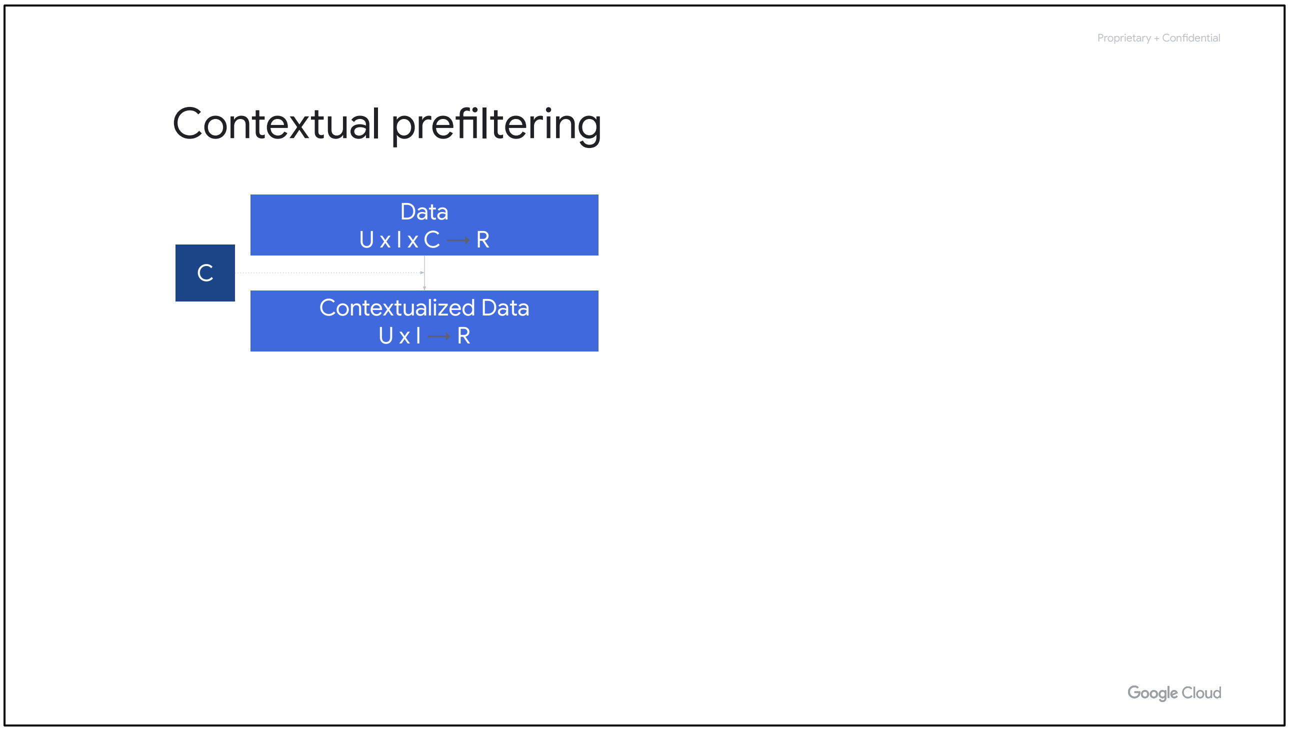
For contextual prefiltering, we start with our user by item by context tensor that contains ratings. In traditional collaborative filtering recommendation systems, we have just a user-item interaction matrix that contains ratings.



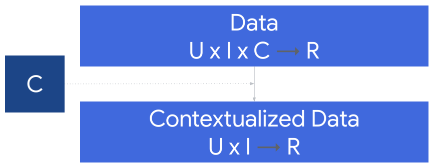
Contextual prefiltering



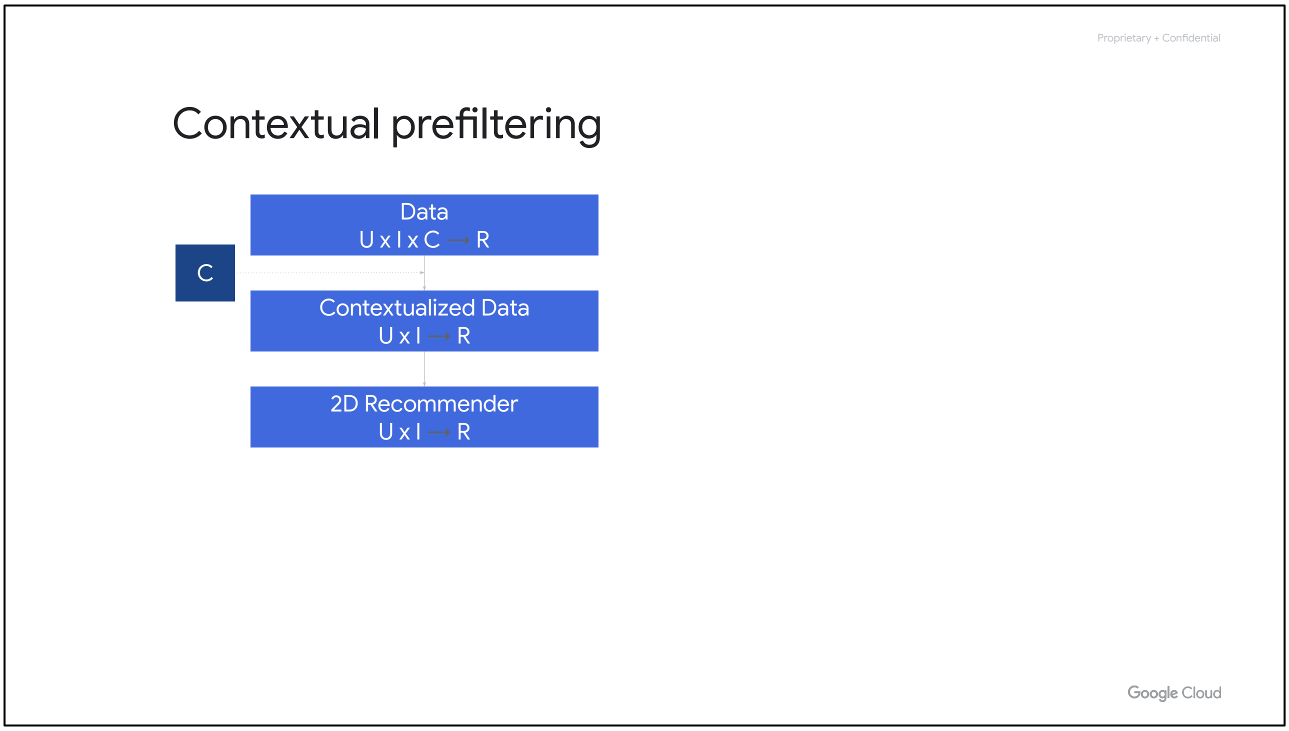
We then apply a filter on our context, before the traditional user-item interaction matrix recommendation system, which is why it is called prefiltering. As an example of this, imagine we are looking at movie theater tickets online, and we hope to go Thursday after work. Therefore, the algorithm will filter out all movies that don't have the time dimension of context equaling Thursday. However, we need to make sure things don't get too specific. For example, this user wanting to go to this exact theater, on Thursday, with a friend from work, where there will be special effects glasses to wear during the movie. That is a really specific context and there are probably going to be sparsity issues. We can also use context generalization, where instead of saying the user wants to see a movie on exactly Thursday, it is a blend of Thursday and weekday, for instance.



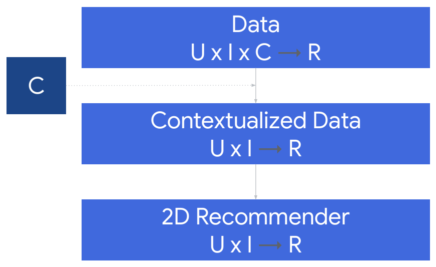
Contextual prefiltering



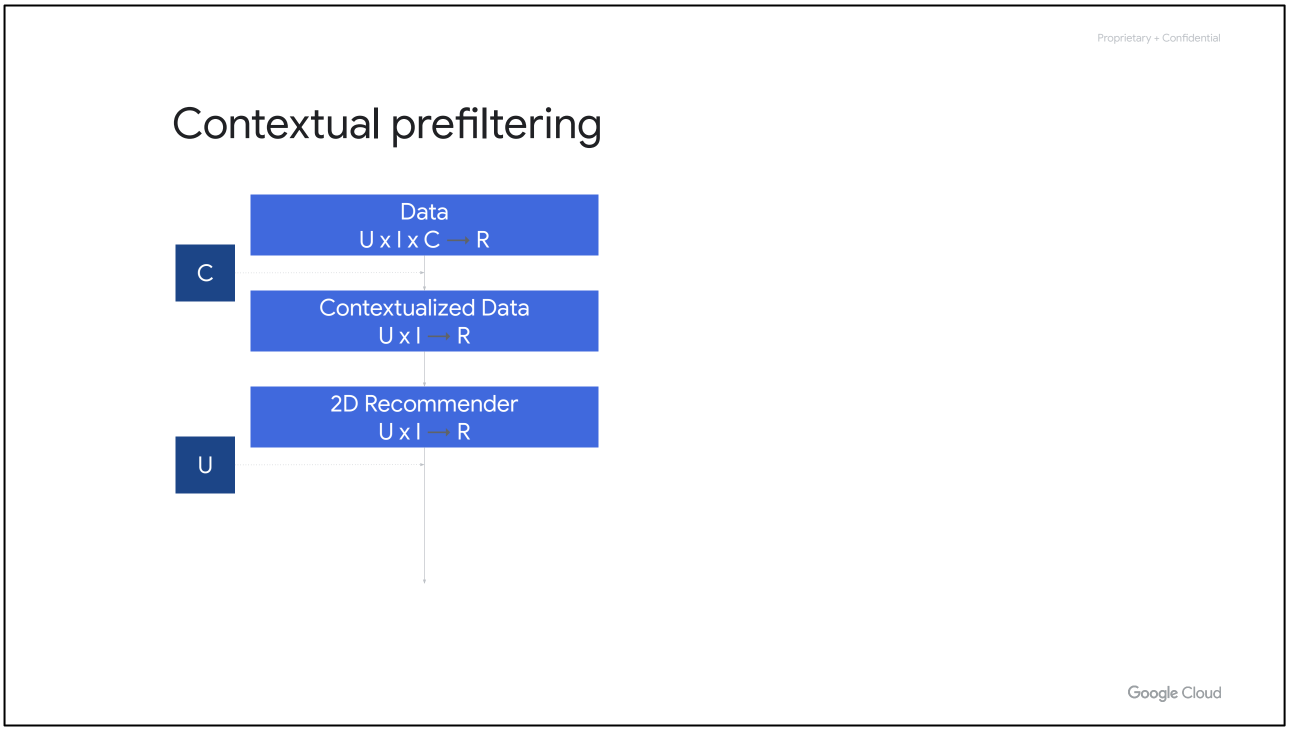
After the data has been filtered by context, the data is once again our usual user-item interaction matrix, which we can use in our traditional collaborative filtering recommendation systems. Note that the contextualized data is filtered by certain dimensions of context, so it is actually a subset of the original data.



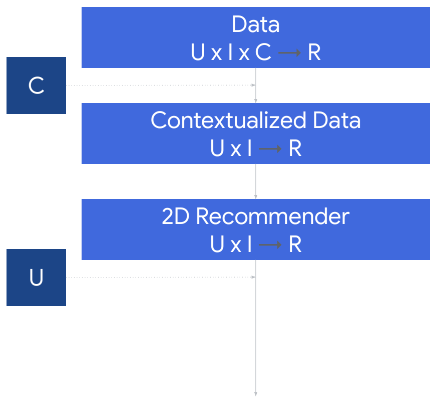
Contextual prefiltering



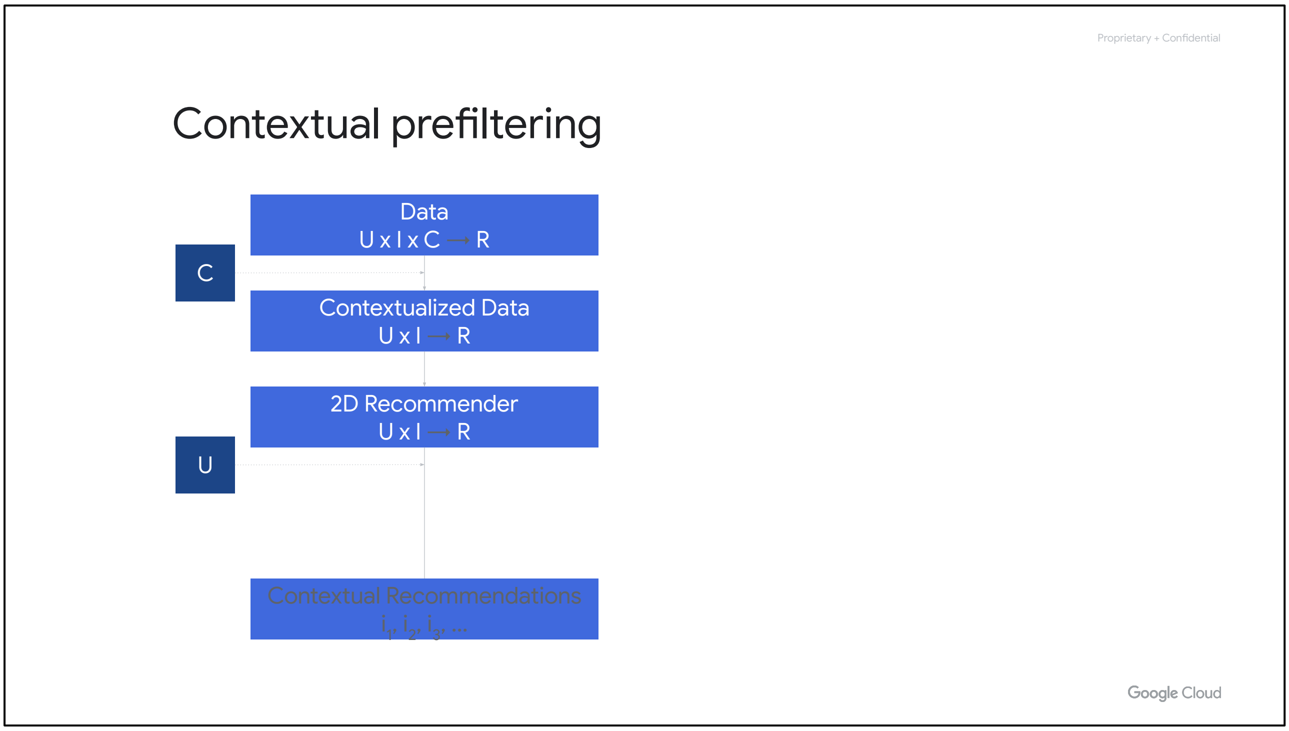
Our contextualized data now goes through a traditional two-dimensional collaborative filtering recommendation system. This is a big plus, because there is no need to develop and implement new algorithms to handle this multi-dimensional context in our input tensor. As we've learned, our recommendation system will simultaneously learn the user and item embeddings that we can then use to make recommendations.



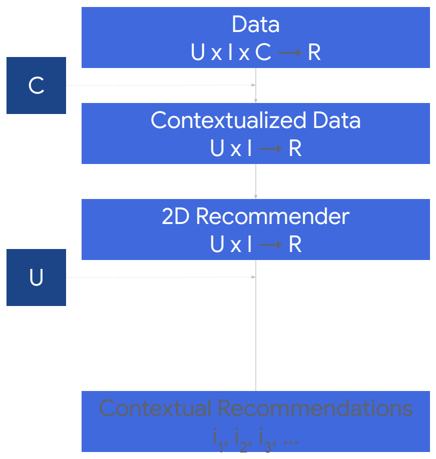
Contextual prefiltering



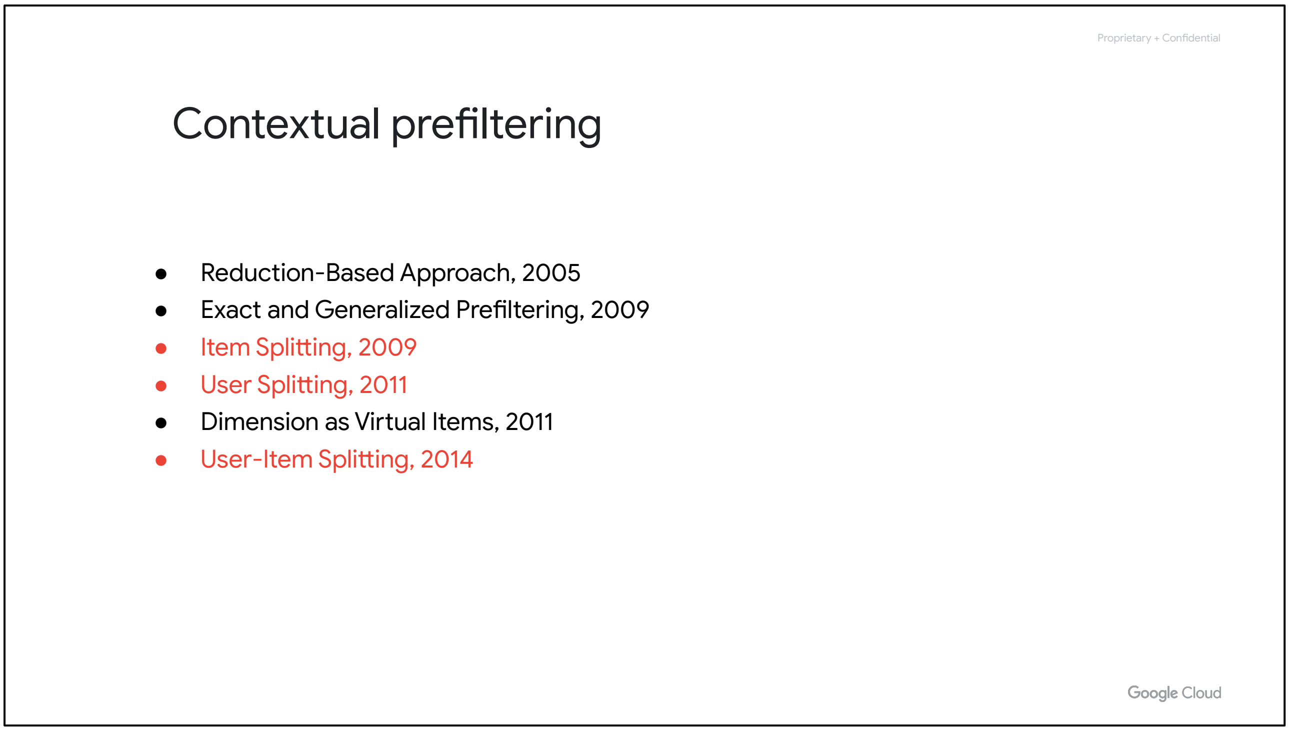
Next, as we learned in module 3, we now apply our user vector to the embeddings learned by the recommender to get a predicted rating for each item the user hasn't already interacted with.



Contextual prefiltering



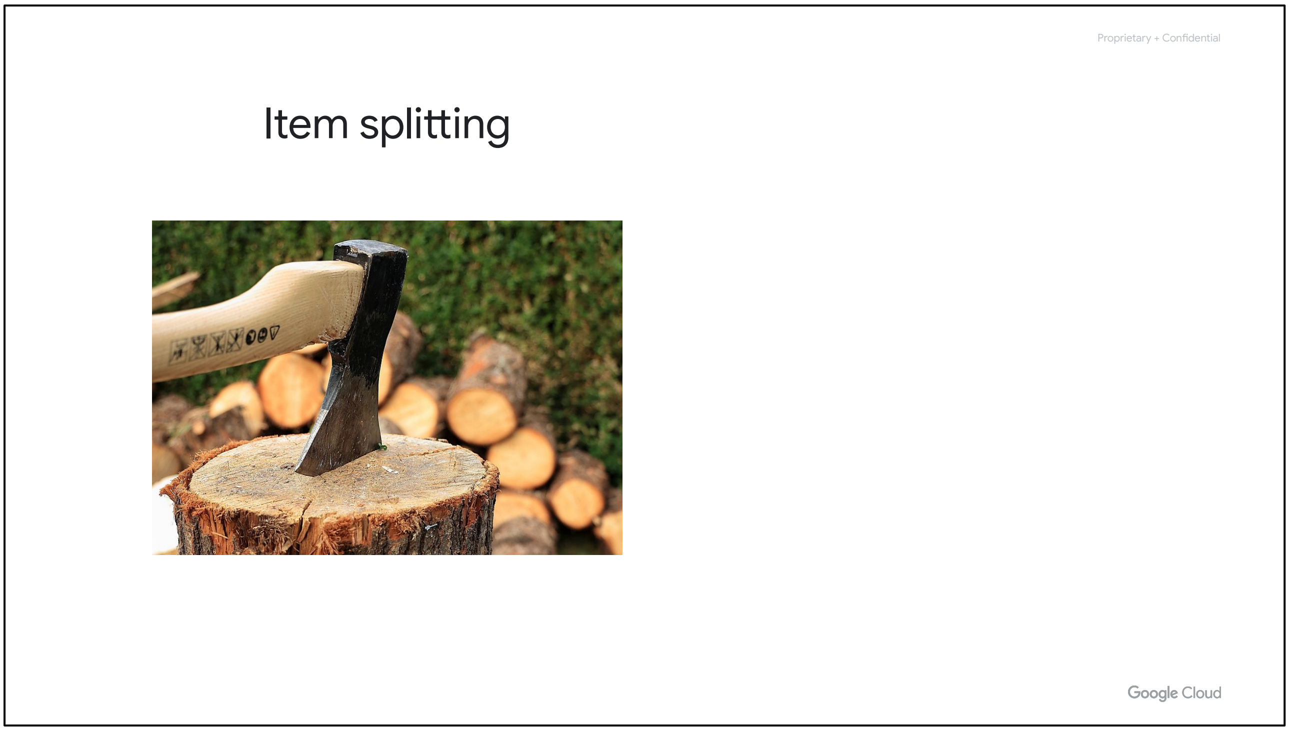
Lastly, for contextual prefiltering, we return the top k item recommendations to the user.



Contextual prefiltering

- Reduction-Based Approach, 2005
- Exact and Generalized Prefiltering, 2009
- Item Splitting, 2009
- User Splitting, 2011
- Dimension as Virtual Items, 2011
- User-Item Splitting, 2014

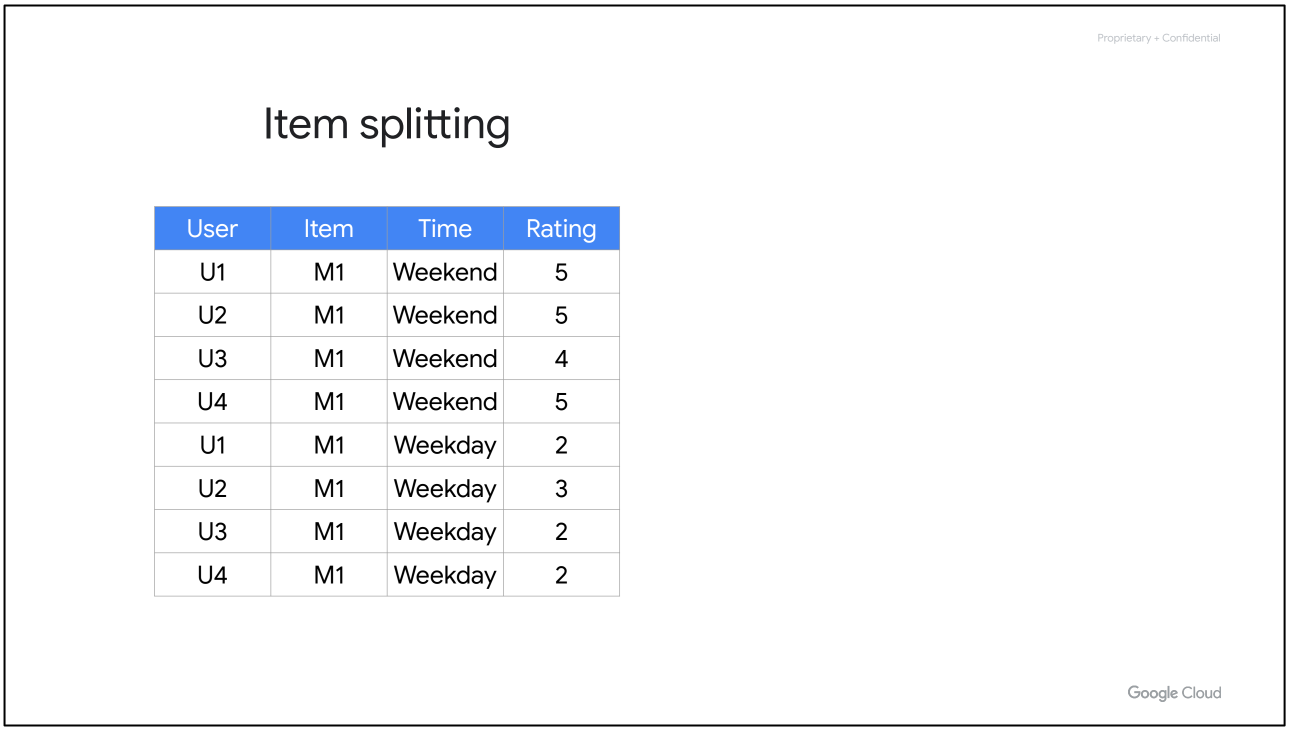
Many different contextual prefiltering algorithms have been developed. Because this is contextual prefiltering, they all are different methods of representing and splitting the data so that it can be used in a traditional two-dimensional recommendation system. We'll take a deeper look at item splitting, user splitting, and the combination of the two with user-item splitting.



Item splitting

We've talked about how context can change our perception and thus sentiment of the same item. Therefore, it isn't unreasonable to come up with a data representation model where we split items into item-context pairs.

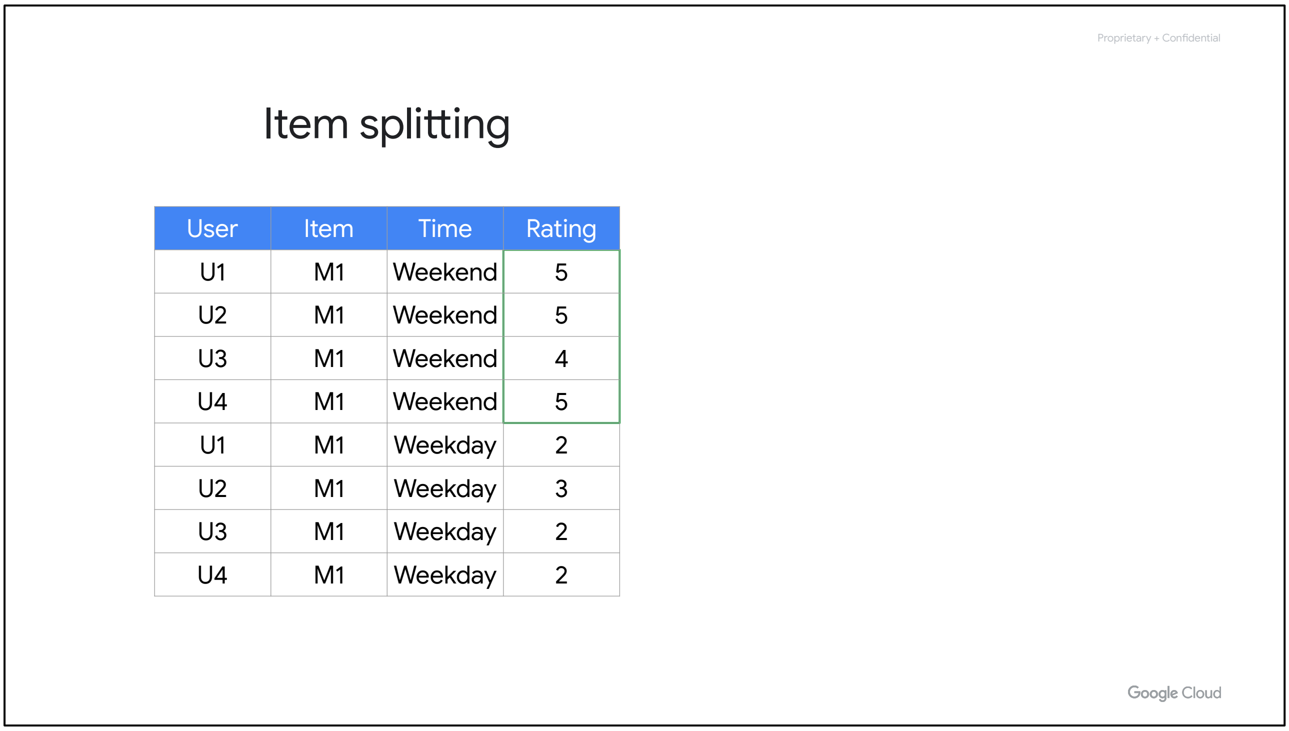
https://pixabay.com/en/axe-wooden-block-wood-chop-firewood-1705787/ cc0



Item splitting

User	Item	Time	Rating
U1	M1	Weekend	5
U2	M1	Weekend	5
U3	M1	Weekend	4
U4	M1	Weekend	5
U1	M1	Weekday	2
U2	M1	Weekday	3
U3	M1	Weekday	2
U4	M1	Weekday	2

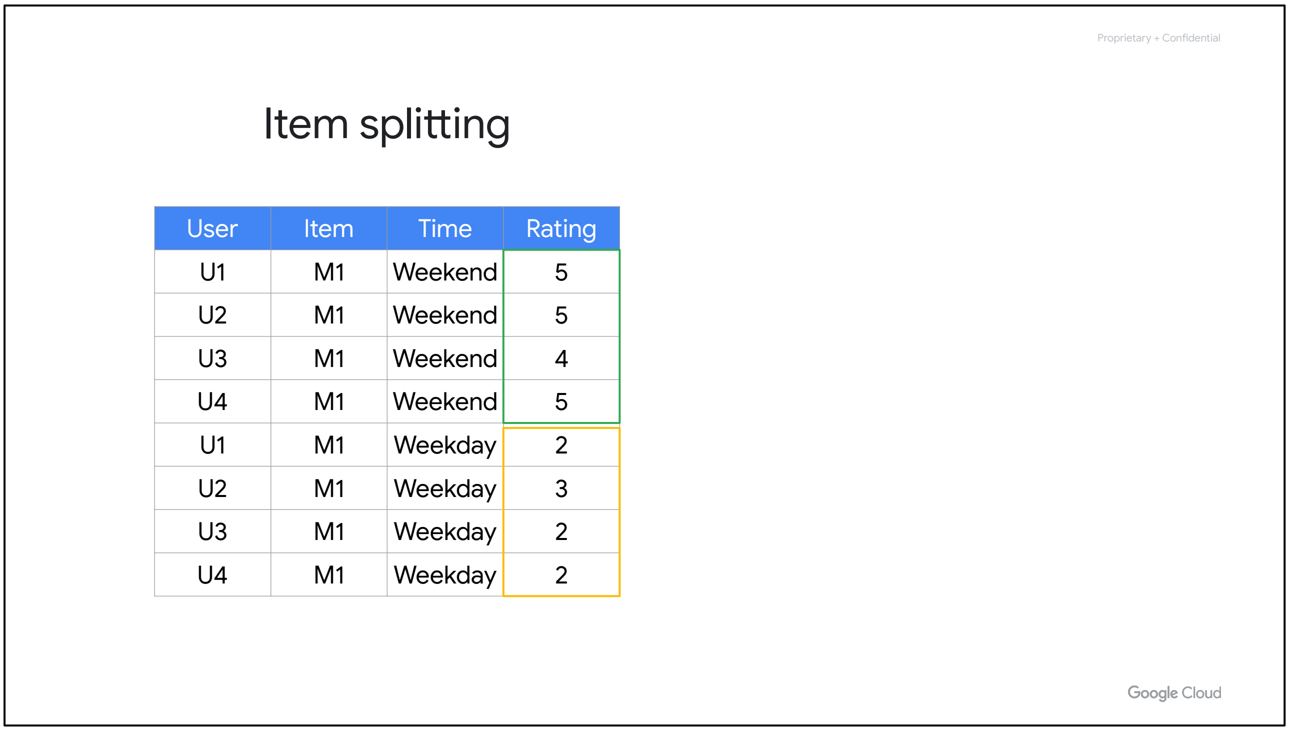
Here is an example of a user-item-context table, with the associated ratings. This example only has one dimension of context, time. Let's briefly analyze it with inspection.



Item splitting

User	Item	Time	Rating
U1	M1	Weekend	5
U2	M1	Weekend	5
U3	M1	Weekend	4
U4	M1	Weekend	5
U1	M1	Weekday	2
U2	M1	Weekday	3
U3	M1	Weekday	2
U4	M1	Weekday	2

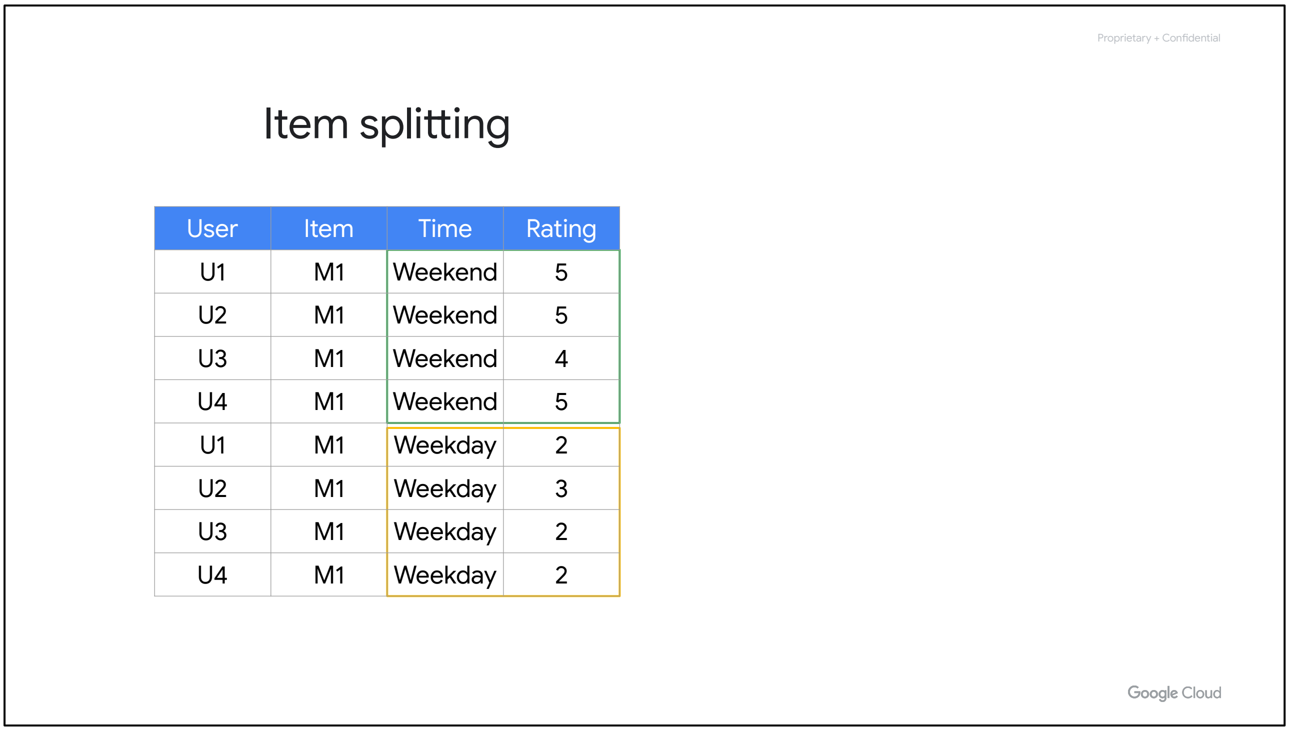
The first thing we might notice is that there are four users and they all watched the same movie twice, but at different times. Looking at the ratings, we can see there is a big block of very high ratings at the top.



Item splitting

User	Item	Time	Rating
U1	M1	Weekend	5
U2	M1	Weekend	5
U3	M1	Weekend	4
U4	M1	Weekend	5
U1	M1	Weekday	2
U2	M1	Weekday	3
U3	M1	Weekday	2
U4	M1	Weekday	2

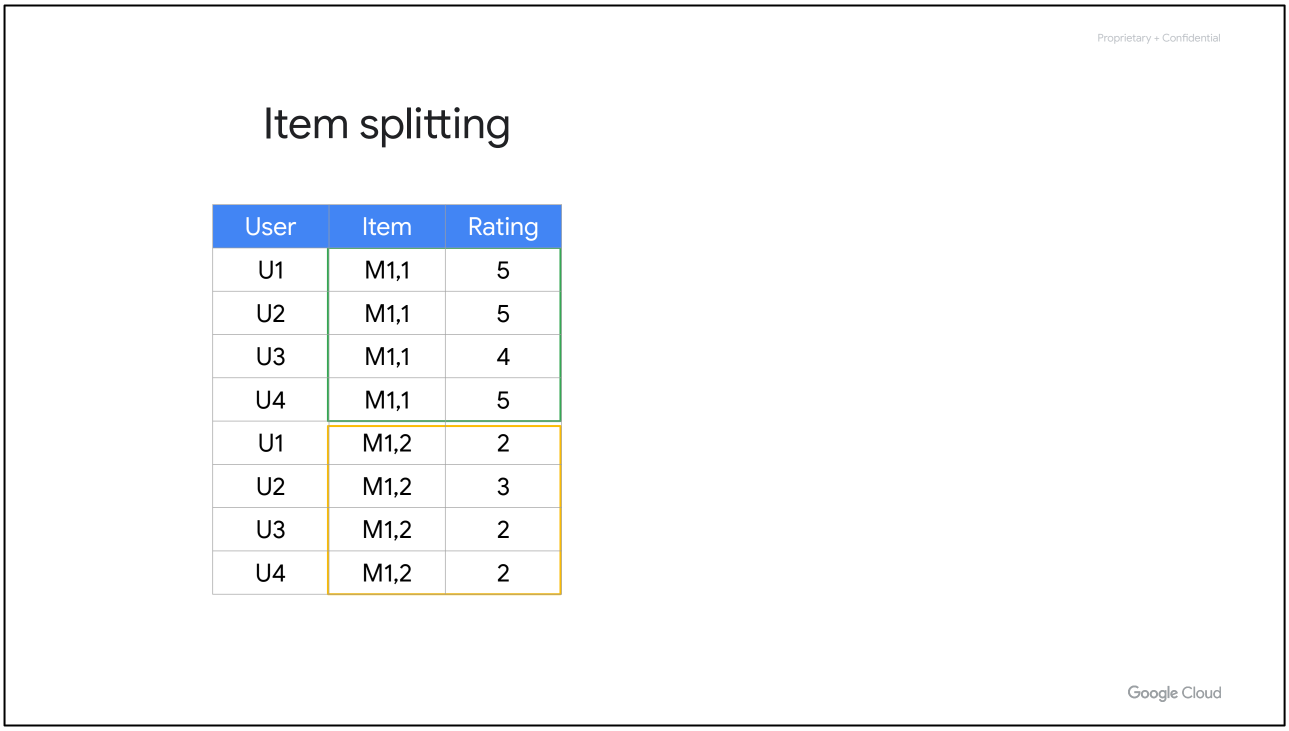
We can also see a large block of low ratings right below. This all looks really strange. What can be causing this? Let's check the context.



Item splitting

User	Item	Time	Rating
U1	M1	Weekend	5
U2	M1	Weekend	5
U3	M1	Weekend	4
U4	M1	Weekend	5
U1	M1	Weekday	2
U2	M1	Weekday	3
U3	M1	Weekday	2
U4	M1	Weekday	2

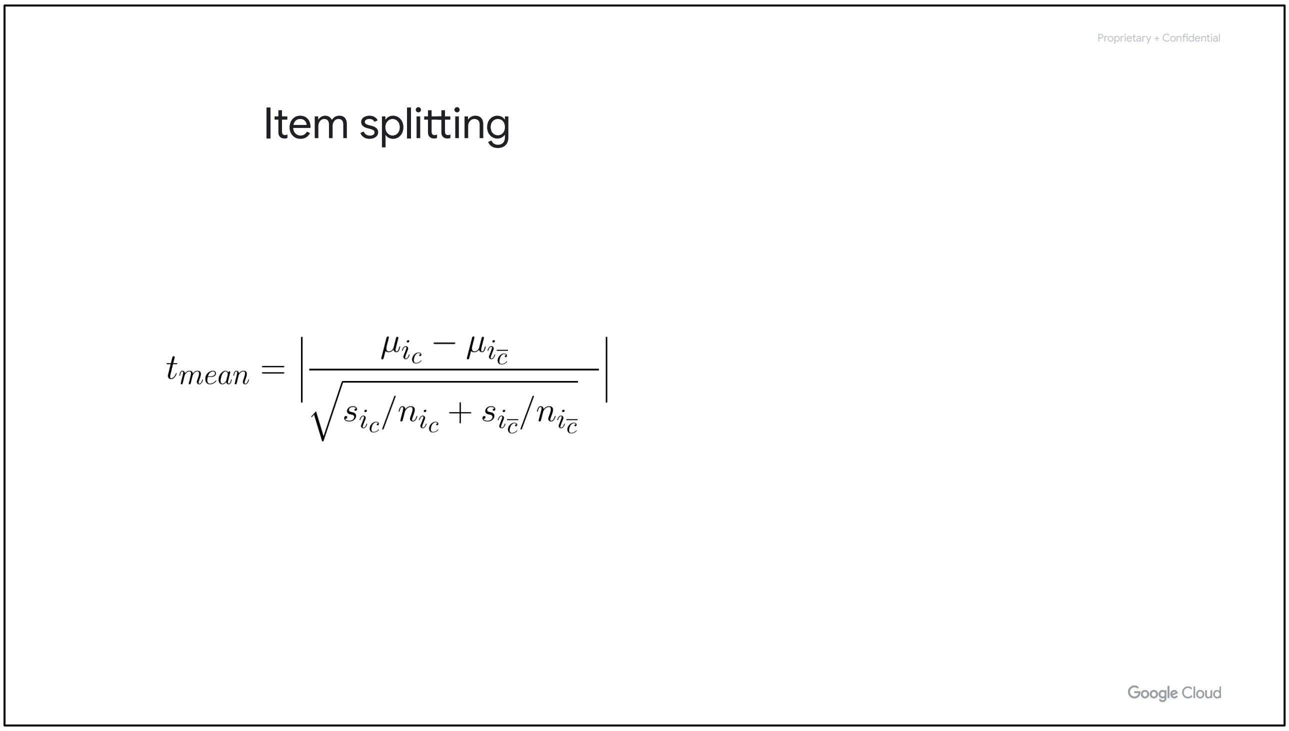
Well there's the difference! All of the high ratings are when the movies were watched on weekends, and all of the low ratings are when the movies were watched on weekdays. Because there is such a significant difference of ratings between the two contexts, with everything else being equal, let's split the item into two item-context entities.



Item splitting

User	Item	Rating
U1	M1,1	5
U2	M1,1	5
U3	M1,1	4
U4	M1,1	5
U1	M1,2	2
U2	M1,2	3
U3	M1,2	2
U4	M1,2	2

There we go, as we can now see, our items have been stratified across context, they are blended together. This functions as if the item was actually multiple items. This is easy to see when we have a small toy dataset like this, but how would we do it for much larger and more complex datasets?

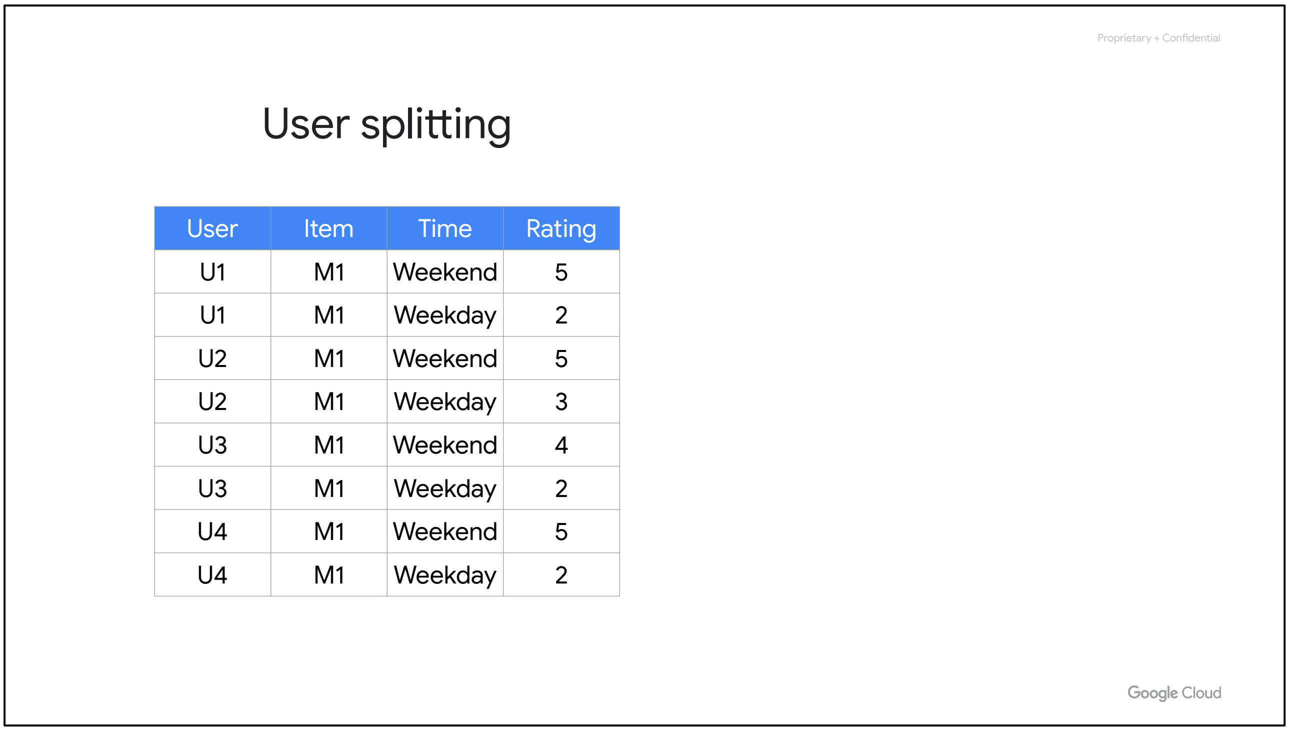


Item splitting

$$t_{mean} = \left| \frac{\mu_{i_c} - \mu_{i_{\overline{c}}}}{\sqrt{s_{i_c}/n_{i_c} + s_{i_{\overline{c}}}/n_{i_{\overline{c}}}}} \right|$$

We simply can use a t-test on two chunks of ratings, and choose what gives the maximum t value, and thus the smallest p-value. There is simple splitting, when splitting across one dimension of context, and complex splitting, when splitting over multiple dimensions of context. Complex splitting can have sparsity issues and can have overfitting problems, so single splitting is often used to avoid these issues.

```
\mu_{i_c}-\mu_{i_\overline{c}}
\sqrt{s_{i_c}/n_{i_c}+s_{i_\overline{c}}}/n_{i_\overline{c}}
t_{mean}=\left | \frac{\ }{\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ } \right |
```



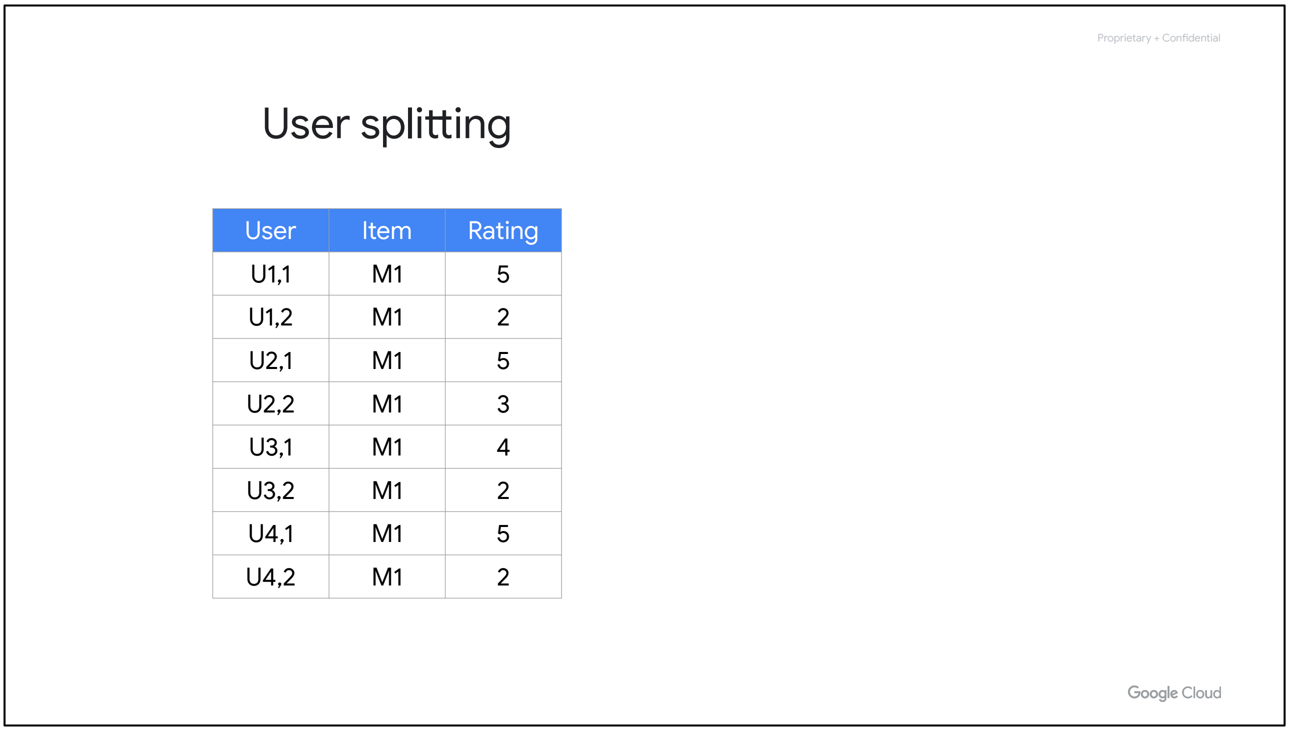
Proprietary + Confidential

User splitting

User	Item	Time	Rating
U1	M1	Weekend	5
U1	M1	Weekday	2
U2	M1	Weekend	5
U2	M1	Weekday	3
U3	M1	Weekend	4
U3	M1	Weekday	2
U4	M1	Weekend	5
U4	M1	Weekday	2

Google Cloud

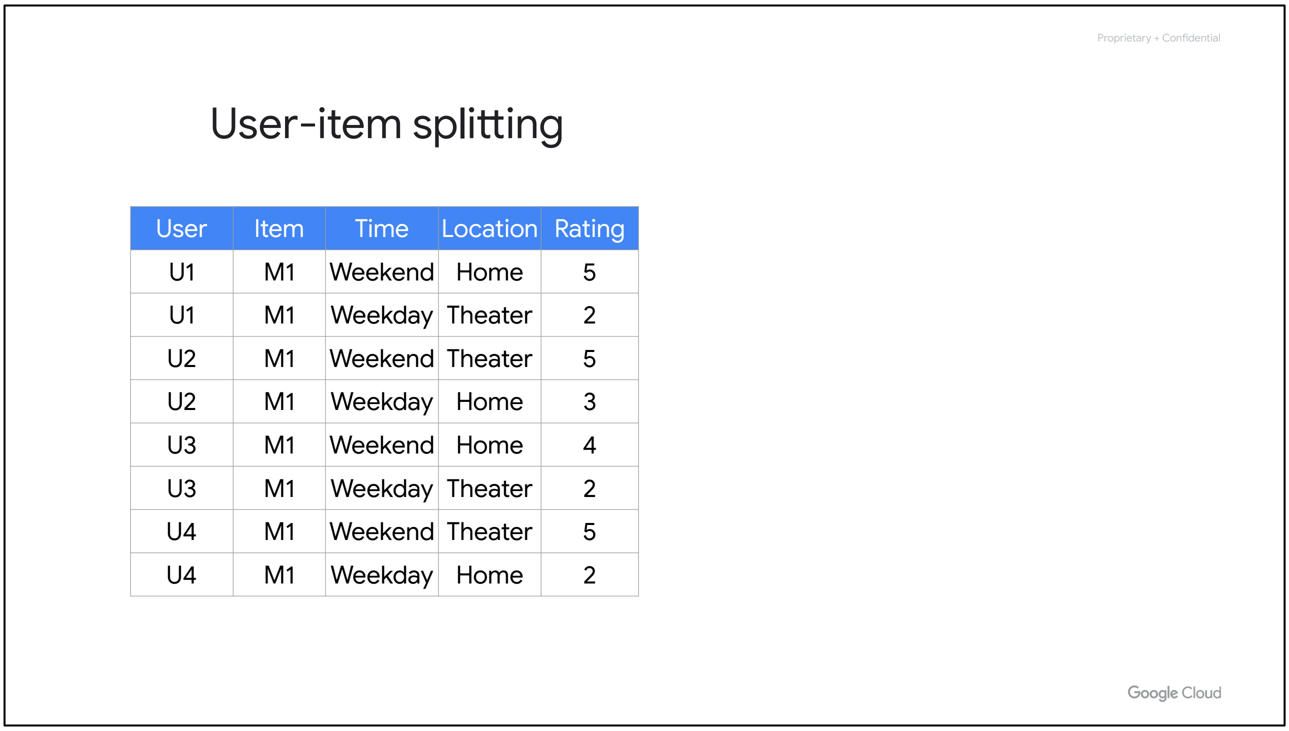
There is also user splitting, which is extremely similar to item splitting, except now we split along user rather than item. If we arrange our data now to be sorted by user, our user, item, context table looks like this.



User splitting

User	Item	Rating
U1,1	M1	5
U1,2	M1	2
U2,1	M1	5
U2,2	M1	3
U3,1	M1	4
U3,2	M1	2
U4,1	M1	5
U4,2	M1	2

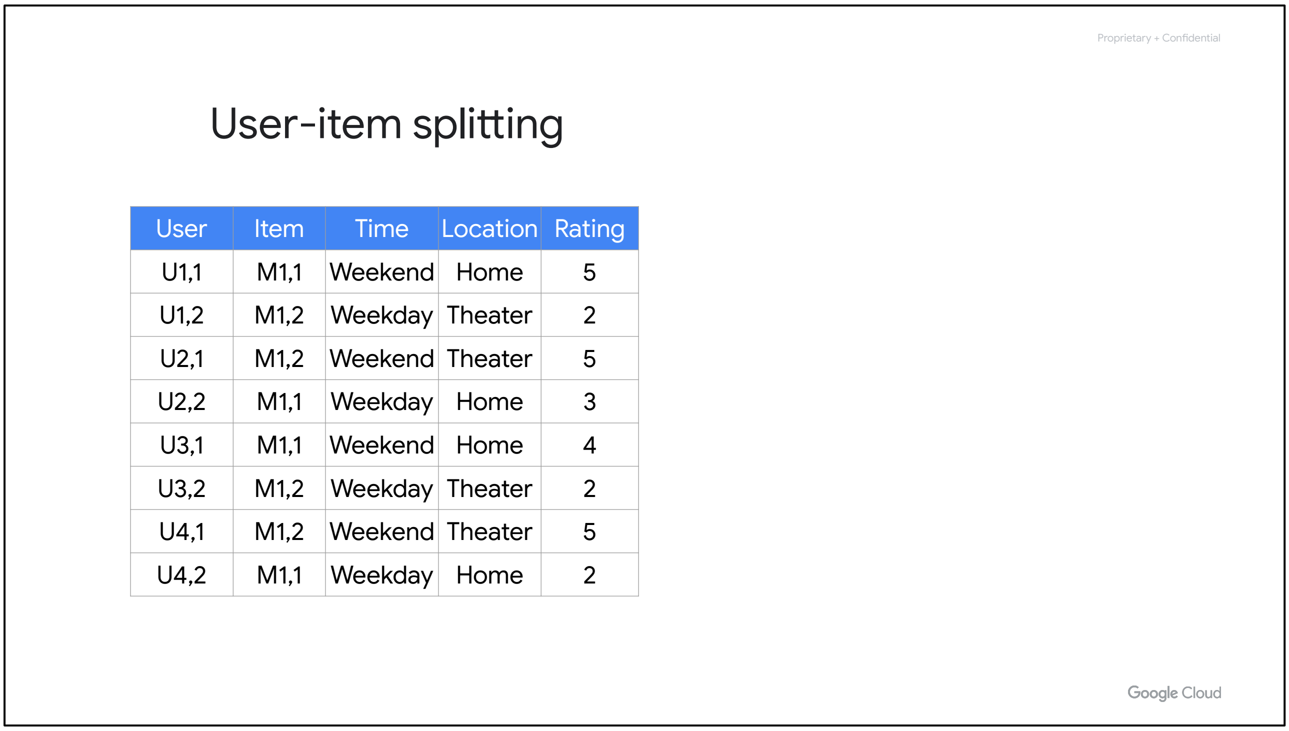
With user splitting, we are now splitting along users as if they are separate users interacting with the item. Users are basically now user-context pairs.



User-item splitting

User	Item	Time	Location	Rating
U1	M1	Weekend	Home	5
U1	M1	Weekday	Theater	2
U2	M1	Weekend	Theater	5
U2	M1	Weekday	Home	3
U3	M1	Weekend	Home	4
U3	M1	Weekday	Theater	2
U4	M1	Weekend	Theater	5
U4	M1	Weekday	Home	2

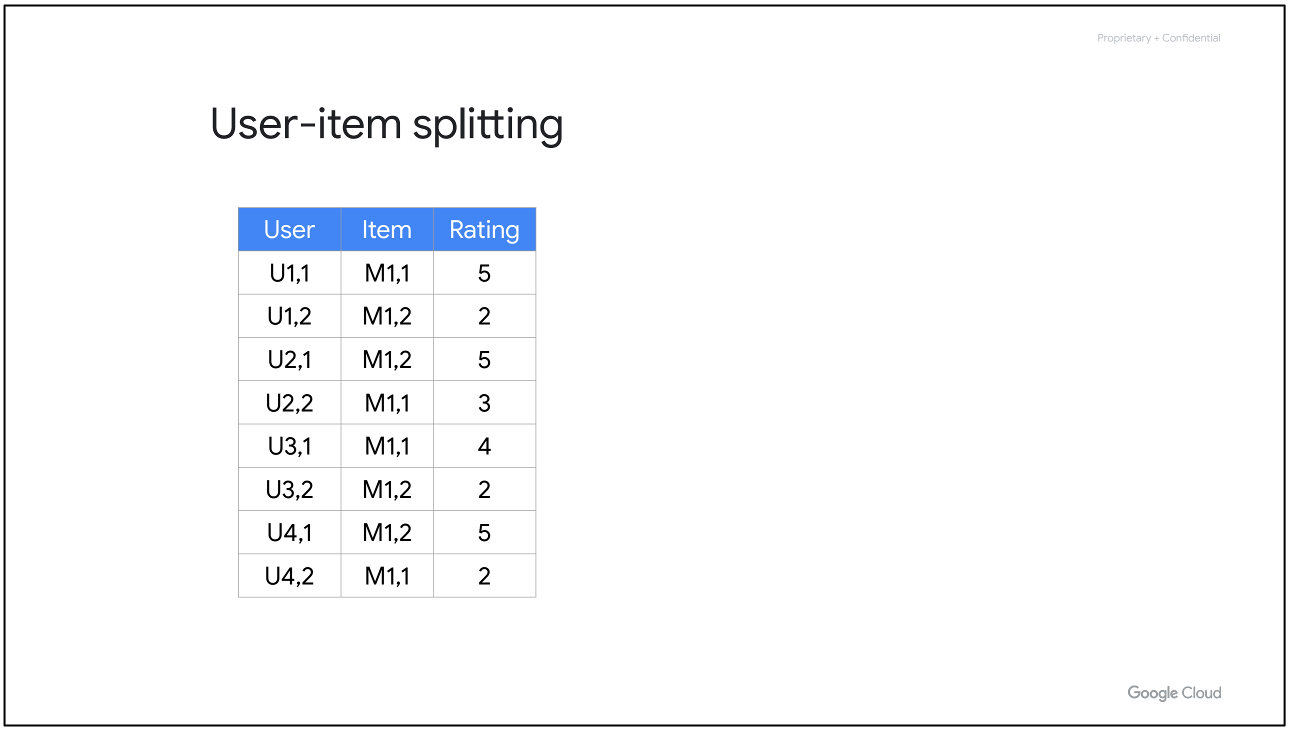
As the name suggests, user item splitting is splitting along both dimensions and blending context into users and items. There is a new dimension of context, location, added to help make this example more clear.



User-item splitting

User	Item	Time	Location	Rating
U1,1	M1,1	Weekend	Home	5
U1,2	M1,2	Weekday	Theater	2
U2,1	M1,2	Weekend	Theater	5
U2,2	M1,1	Weekday	Home	3
U3,1	M1,1	Weekend	Home	4
U3,2	M1,2	Weekday	Theater	2
U4,1	M1,2	Weekend	Theater	5
U4,2	M1,1	Weekday	Home	2

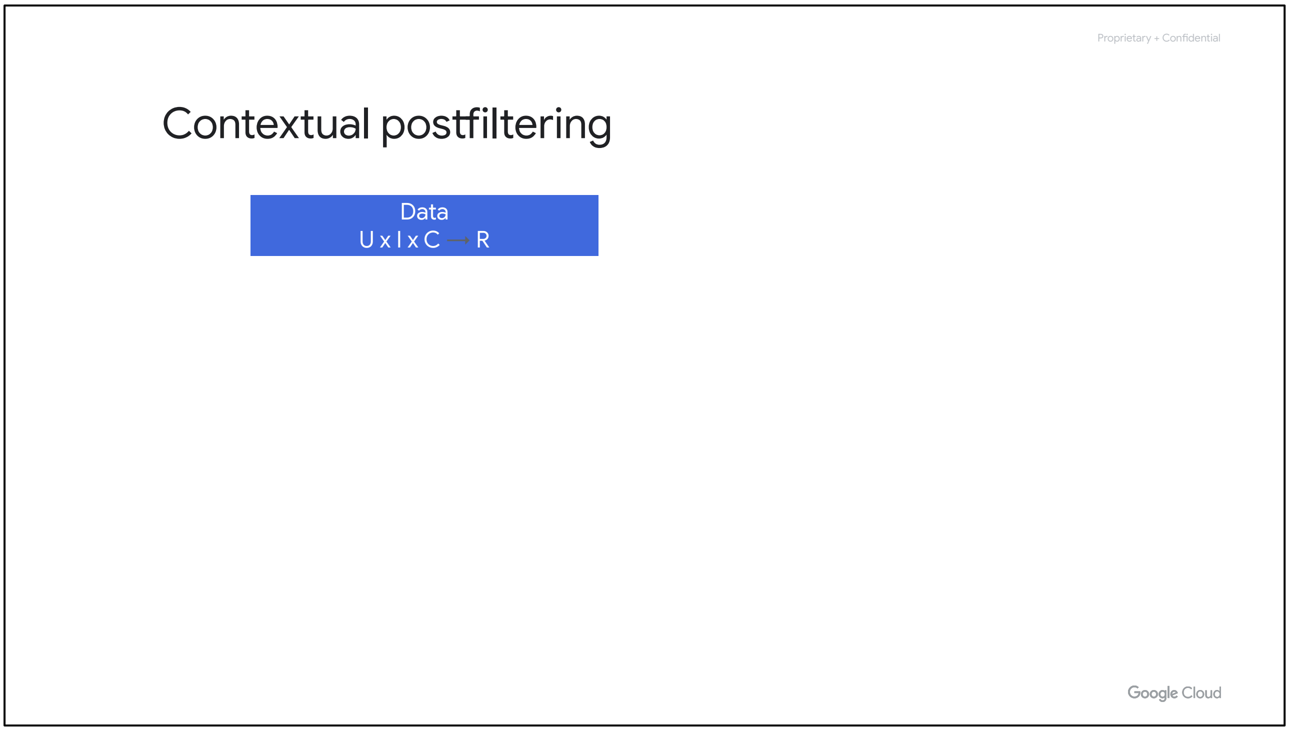
To see what we have done with the splitting, the context columns are still remaining. Here we did two simple splits: a user split, along the time context dimension, and an item split, along the location context dimension.



User-item splitting

User	Item	Rating
U1,1	M1,1	5
U1,2	M1,2	2
U2,1	M1,2	5
U2,2	M1,1	3
U3,1	M1,1	4
U3,2	M1,2	2
U4,1	M1,2	5
U4,2	M1,1	2

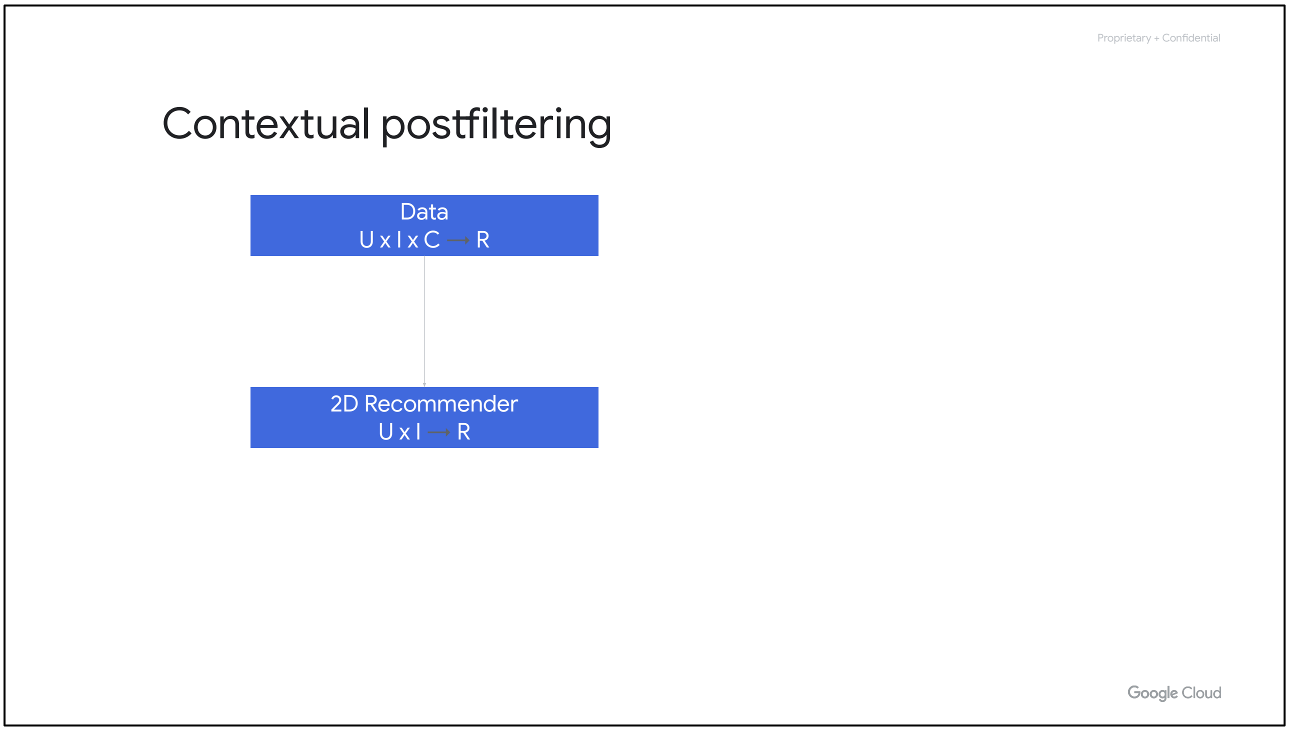
Removing the context columns we split along simplifies the table to this, where we have user-time context pairs as individual users and item-location context pairs as individual items. We can now send this contextually prefiltered data to our traditional, two-dimensional recommendation system.



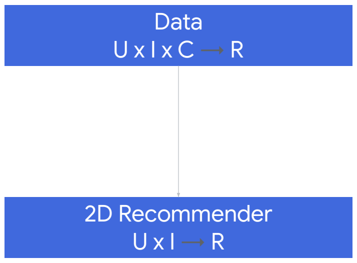
Contextual postfiltering

Data
$U \times I \times C \rightarrow R$

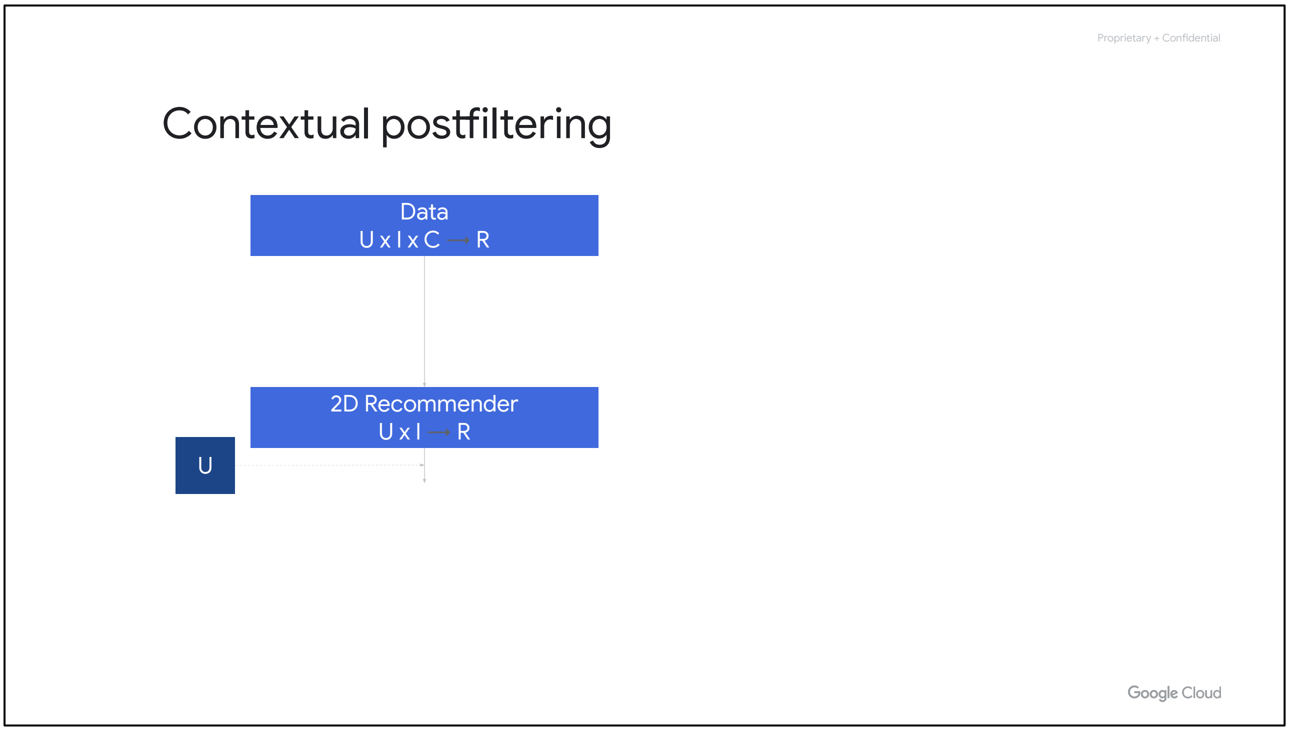
Just as in contextual prefiltering, the contextual postfiltering algorithm begins with our initial user by item by multi-dimensional context tensor, containing ratings.



Contextual postfiltering

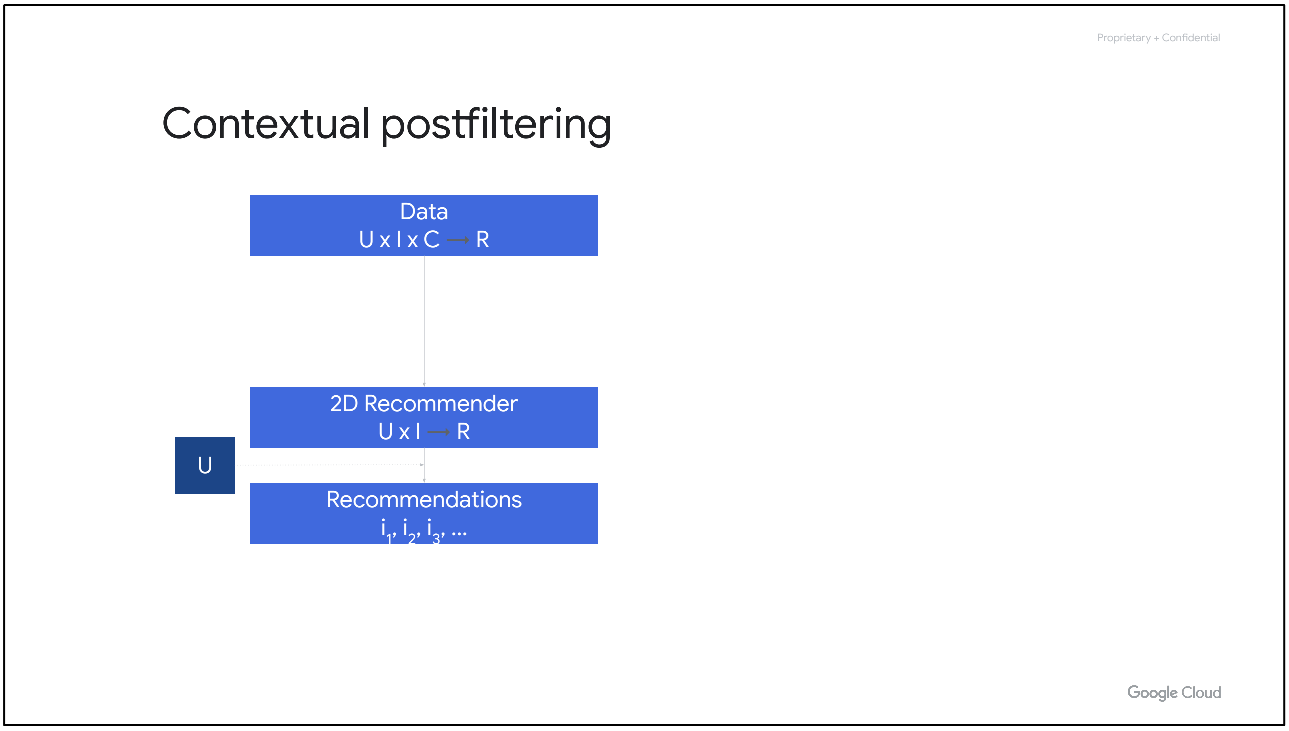


However, notice here, we are sending our initial data directly to our traditional two-dimensional recommendation system. How can this be? Why did we not apply the context filter before the recommender? What happens to all of the context dimensions? Well, we simply ignore them; we ignore context. We process the data as if it was just a user-item interaction matrix. This is good in the respect that we can use our traditional collaborative filtering technology, but aren't the recommendations going to be the same, as if we never had context data to begin with? Let's wait and see to find out.



Contextual postfiltering

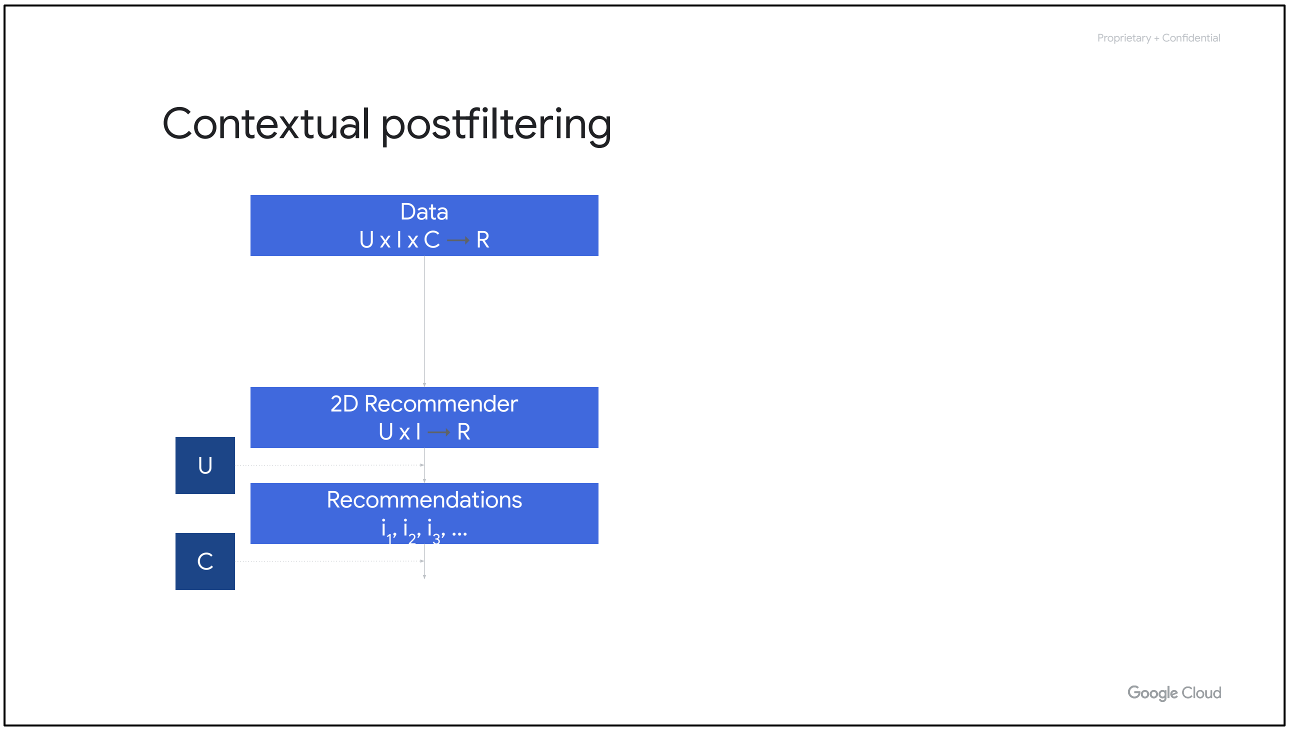
We then apply our user's vector to the output embeddings to get their representation in embedding space.



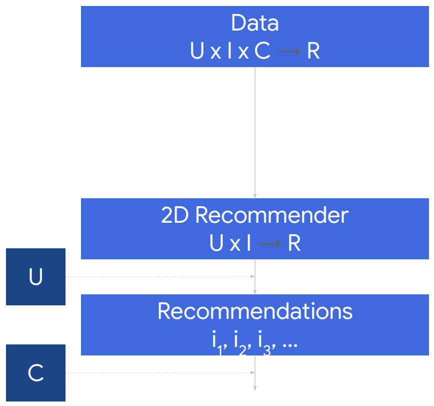
Contextual postfiltering

Data
$U \times I \times C \rightarrow R$
2D Recommender
$U \times I \rightarrow R$
U
Recommendations
$i_1, i_2, i_3, \ldots$

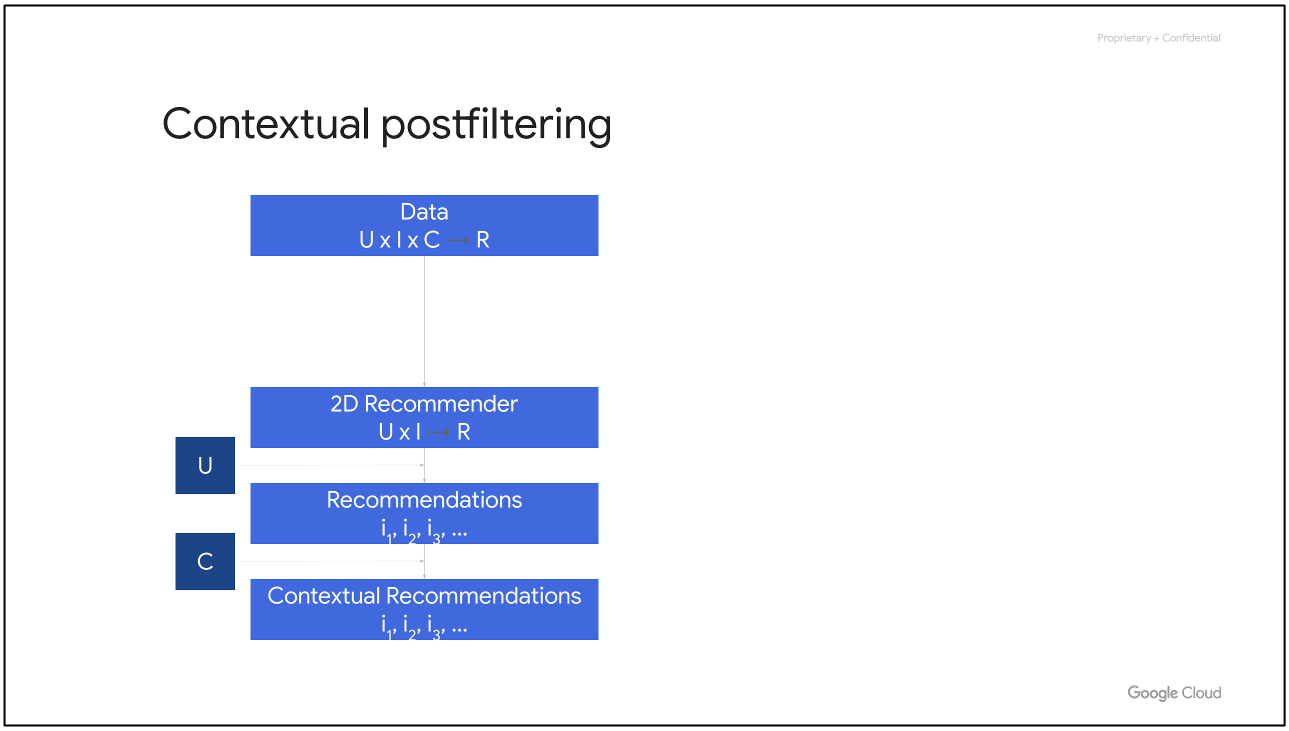
This gives us our recommendations. But these are exactly the same as if we never had context data. Because in machine learning it is such a shame to throw out possibly good data, how do we fix this problem?



Contextual postfiltering

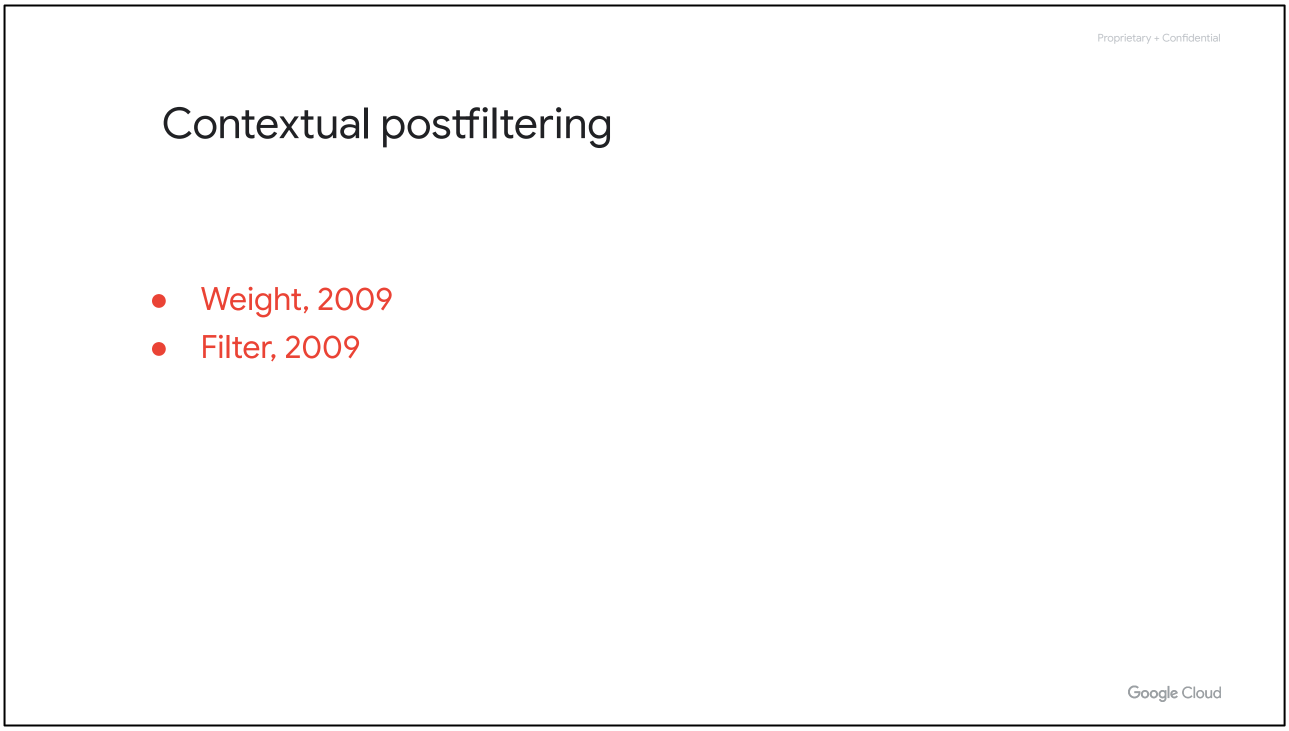


Well we can try to adjust our non-contextual recommendations by applying the context back in. As you can see, this happens after the recommender, hence it is called contextual POST-filtering. This can be done by filtering out recommendations that don't match the current context, or altering the ranking of the predictions, or recommendations, returned by the recommender. These adjustments can be determined using either heuristic or model-based approaches. For example, if our user from before still wants to see a movie on Thursday after work, and on Thursday they usually watch action movies, our postfiltering can filter out all non-action movies from the recommendations returned by our non-contextual recommender.



Contextual postfiltering

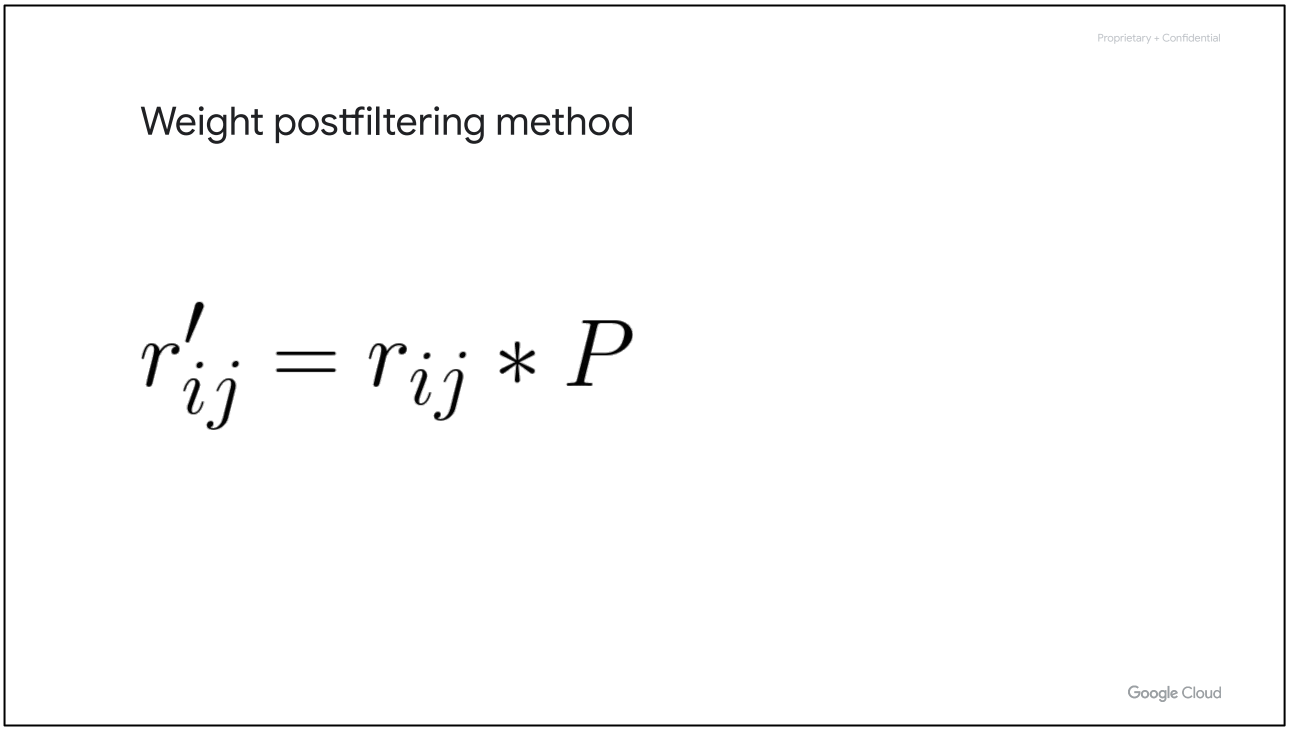
This then gets us finally to the contextual recommendations that we wanted.



Contextual postfiltering

- Weight, 2009
- Filter, 2009

Contextual postfiltering has several methods, of which weight and filter are the most popular and are based on adjusting the non-contextual recommendations, based on the context's relevance to the user. To find this relevance, the contextual probability is calculated for user i, choosing item j, in context c. This is calculated by dividing the number of users similar to user i, who chose the same item j, in the same context c, BY the total number of users similar to user i.

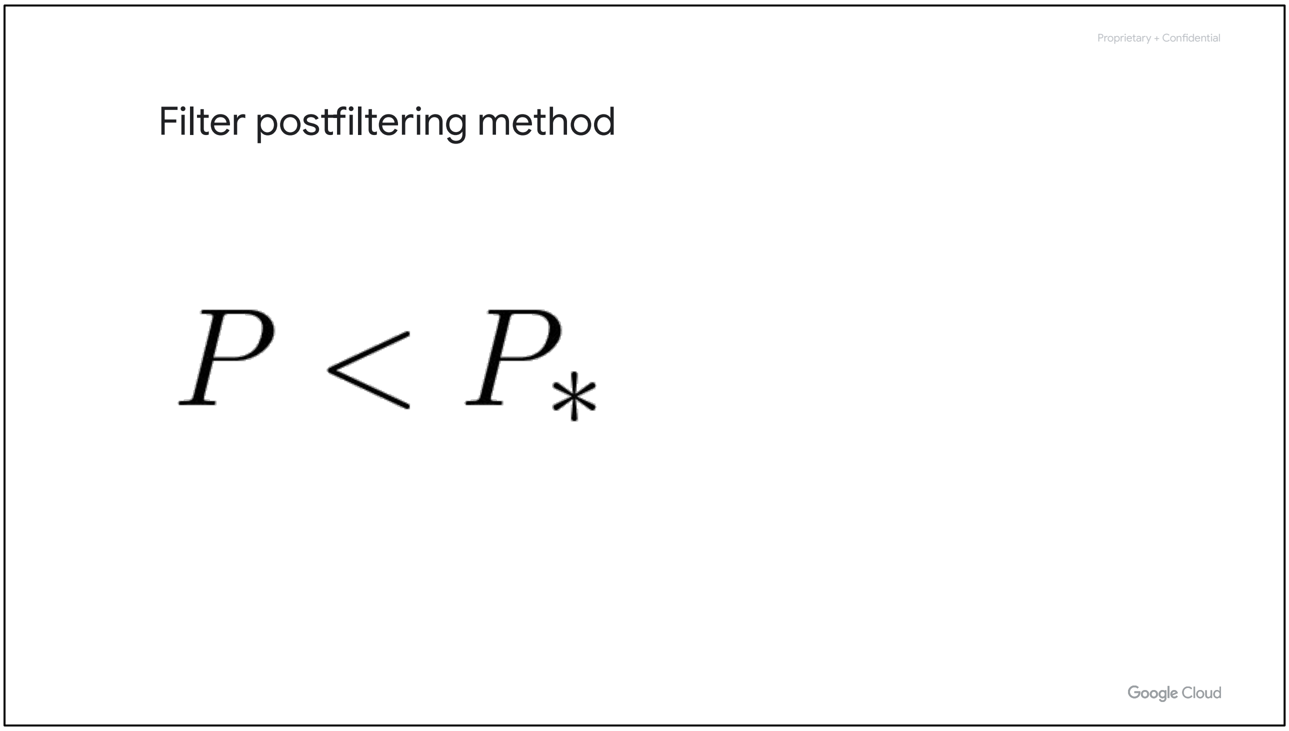


Weight postfiltering method

$$r'_{ij} = r_{ij} * P$$

The weight method multiplies the non-contextualized ratings by the contextual probability to get the adjusted contextualized ratings.

r_{ij}' = r_{ij}*P

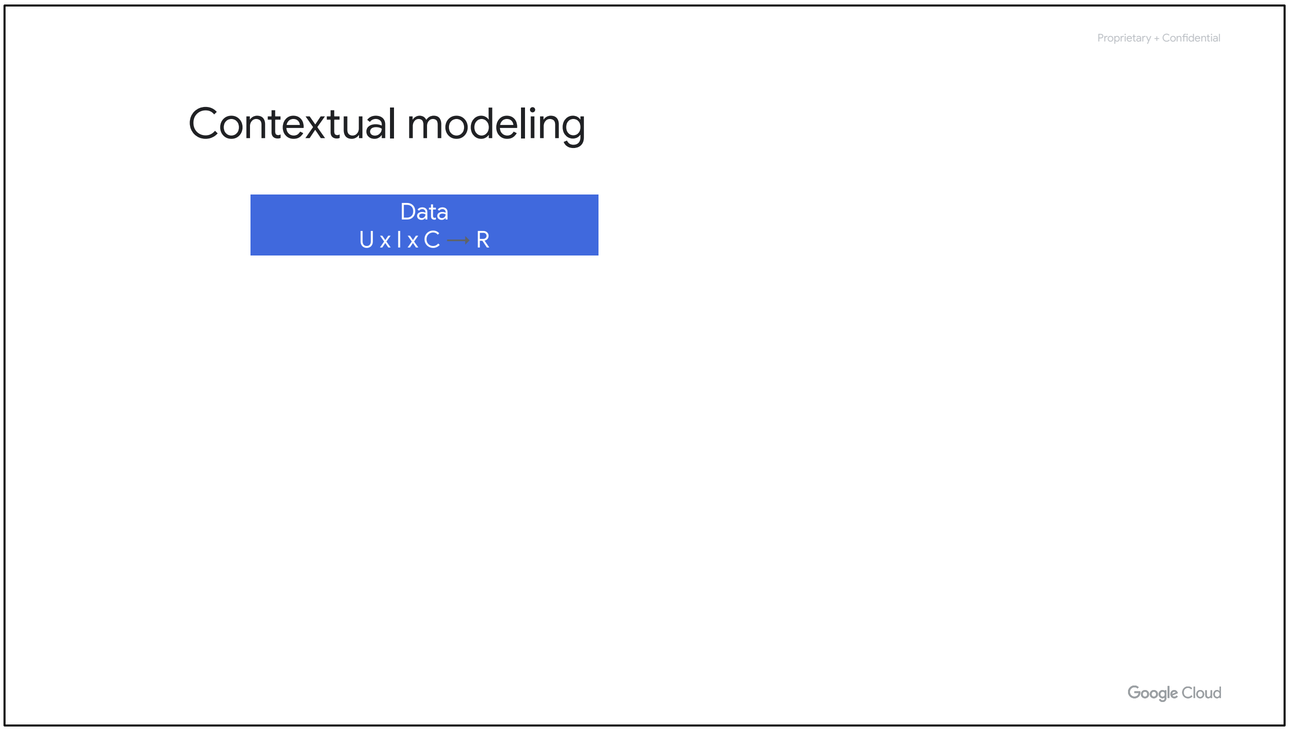


Filter postfiltering method

$$P < P_*$$

The filter method, on the other hand, filters out predicted ratings below a certain threshold value of the contextual probability.

P < P_*

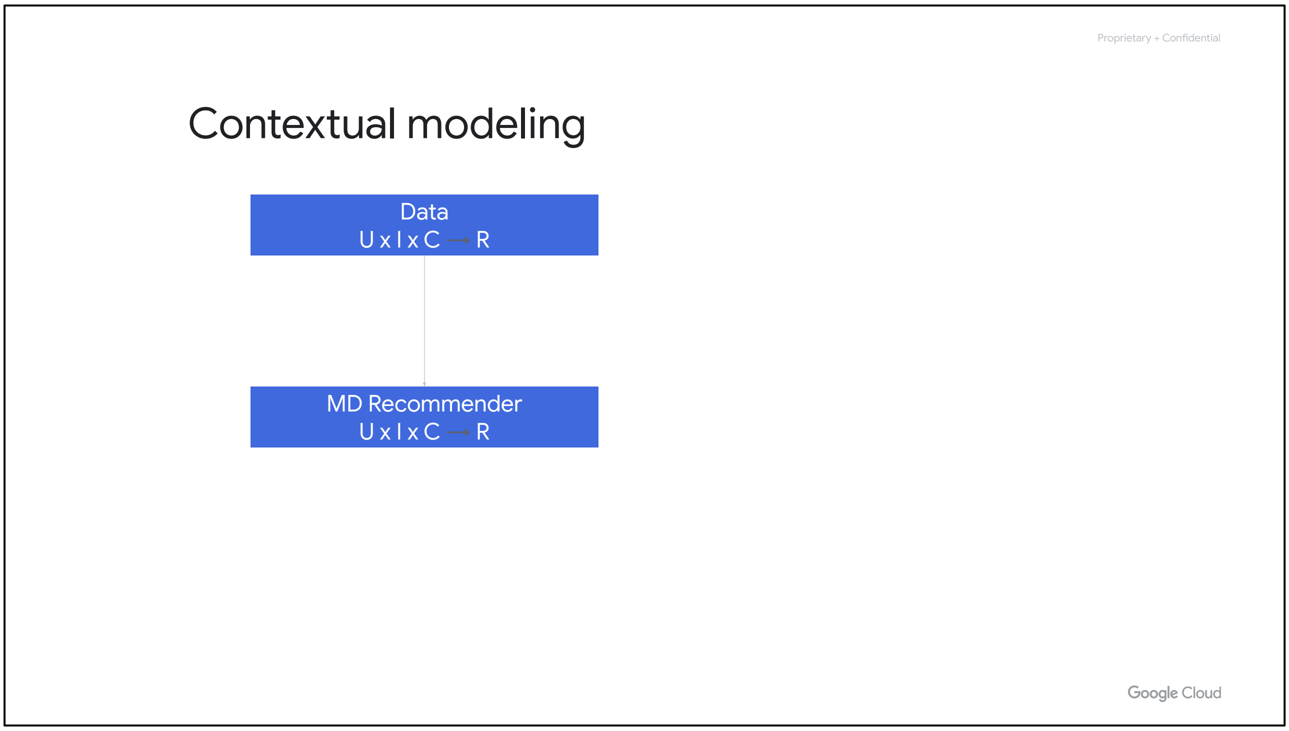


Contextual modeling

Data

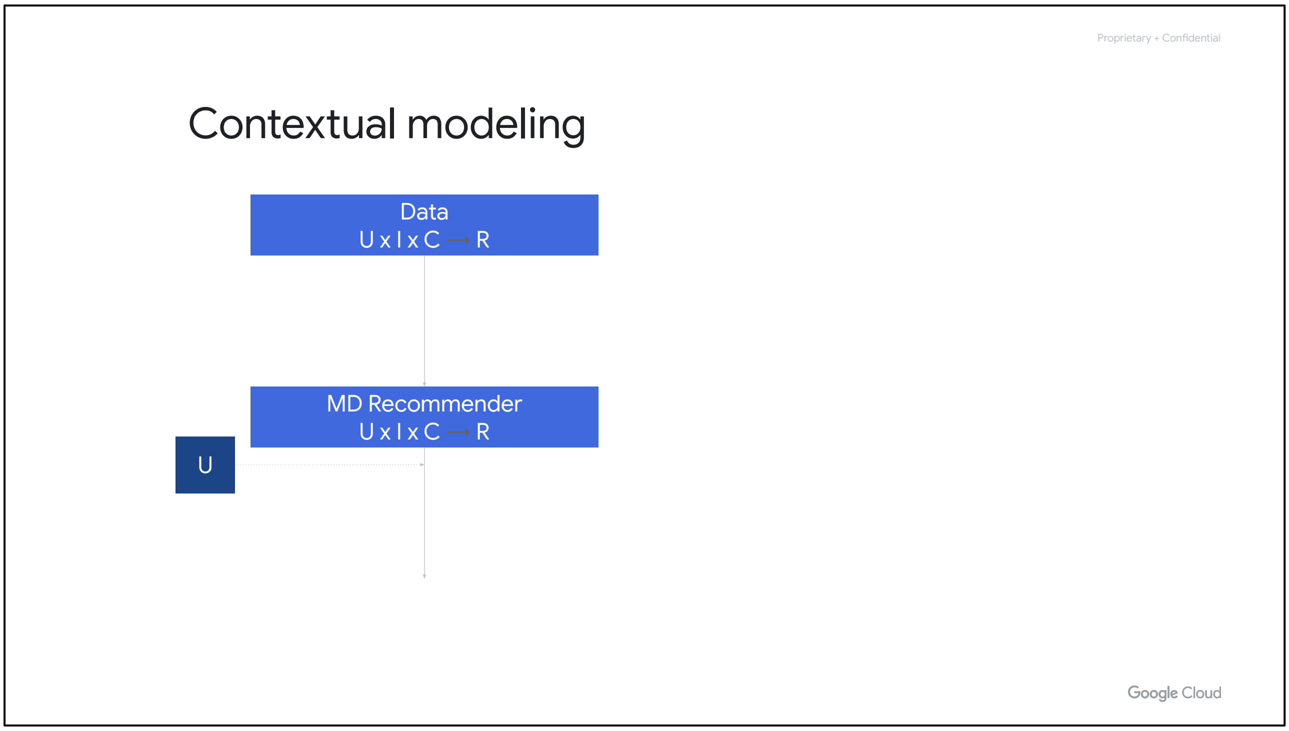
$U \times I \times C \rightarrow R$

Just like the other two types of content-aware algorithms, contextual modeling begins with our user by item by multi-dimensional context tensor, containing ratings.

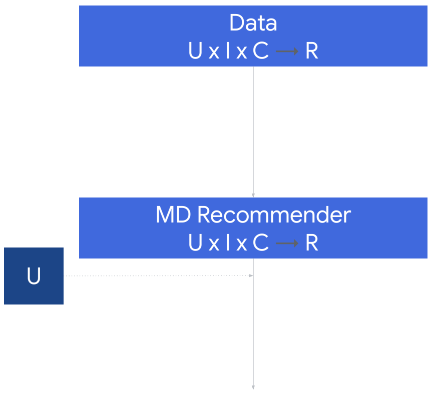


Contextual modeling

This data goes directly to our recommender, but notice this is not our traditional, two-dimensional recommendation system, but an m-dimensional one where context is a part of our model. Context is an explicit predictor, along with the usual user-item relationship. These multi-dimensional recommendation functions can be represented via heuristics or predictive model-based approaches, such as decision trees, regression models, or probabilistic models.

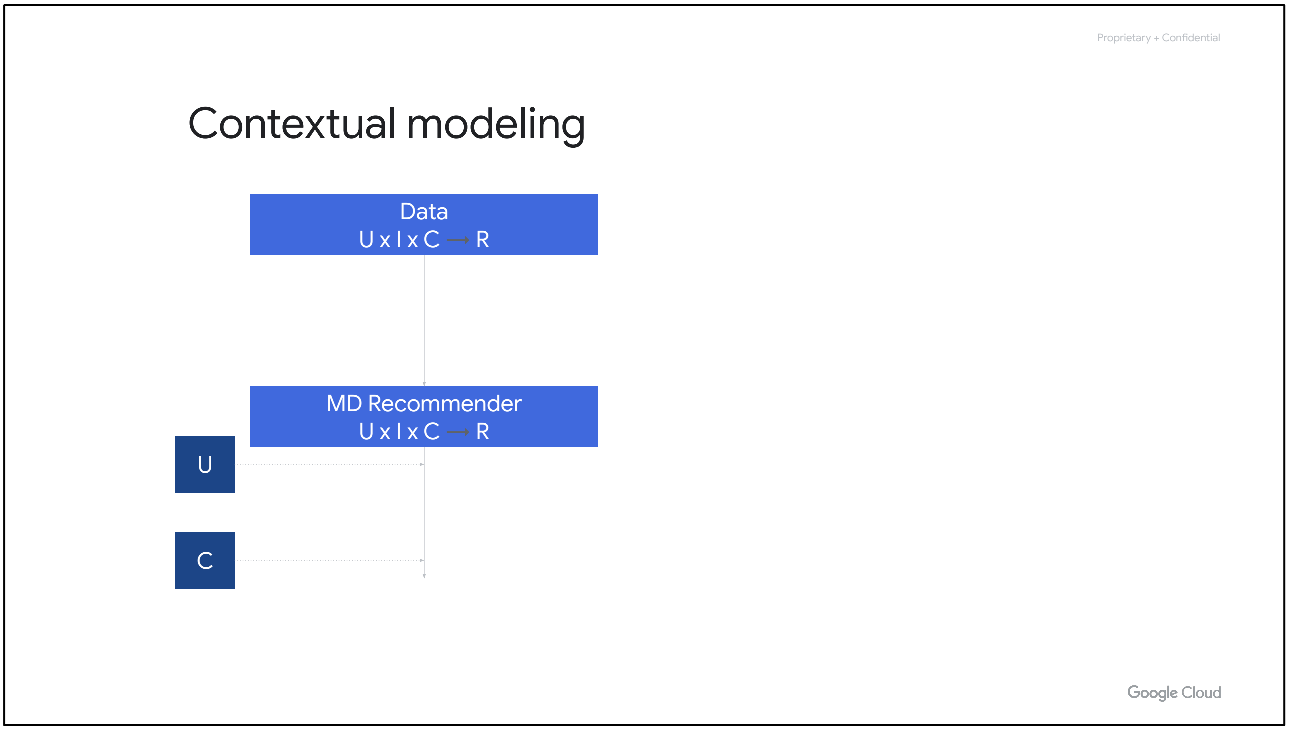

Proprietary + Confidential

Contextual modeling

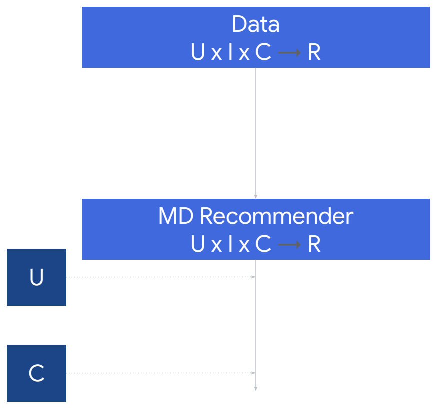


Google Cloud

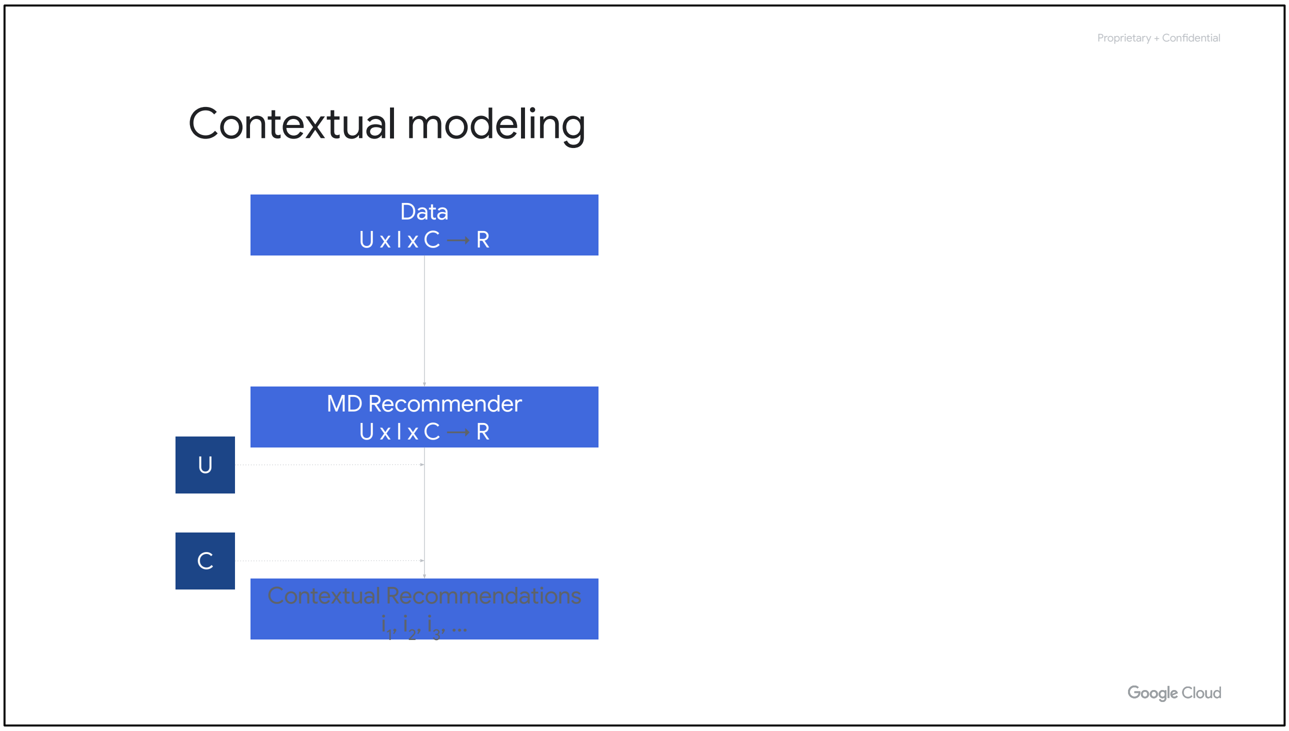
We then can apply our user vector to our multi-dimensional recommendations to get their representation in multi-dimensional embedding space.



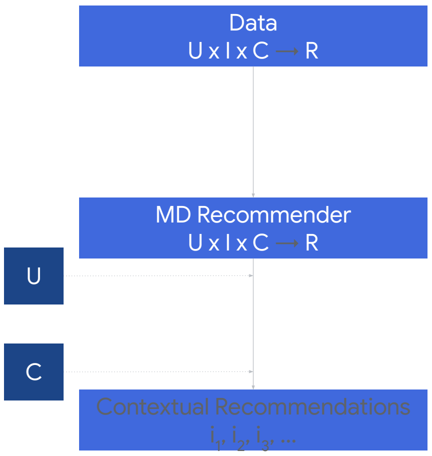
Contextual modeling



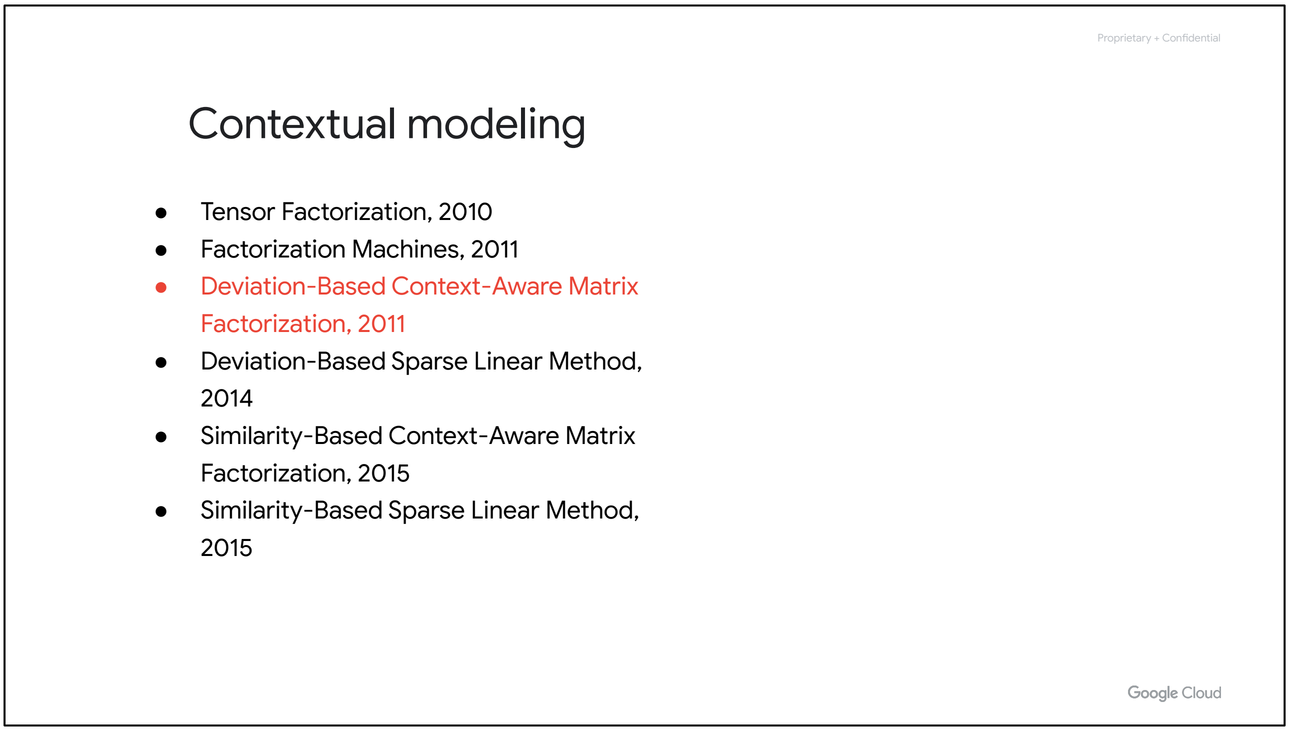
Then we apply our context vector to our multi-dimensional recommendations.



Contextual modeling



Giving us finally our contextual recommendations for our user.

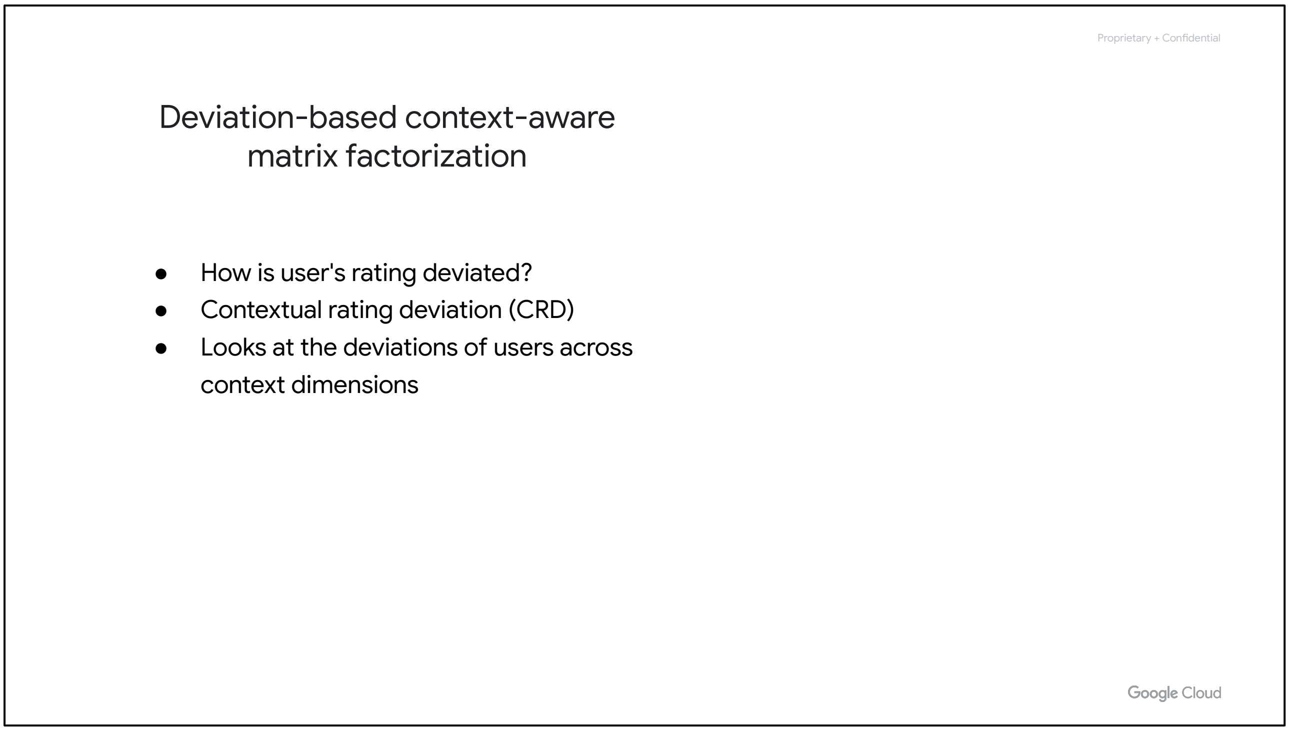

Proprietary + Confidential

Contextual modeling

- Tensor Factorization, 2010
- Factorization Machines, 2011
- Deviation-Based Context-Aware Matrix Factorization, 2011
- Deviation-Based Sparse Linear Method, 2014
- Similarity-Based Context-Aware Matrix Factorization, 2015
- Similarity-Based Sparse Linear Method, 2015

Google Cloud

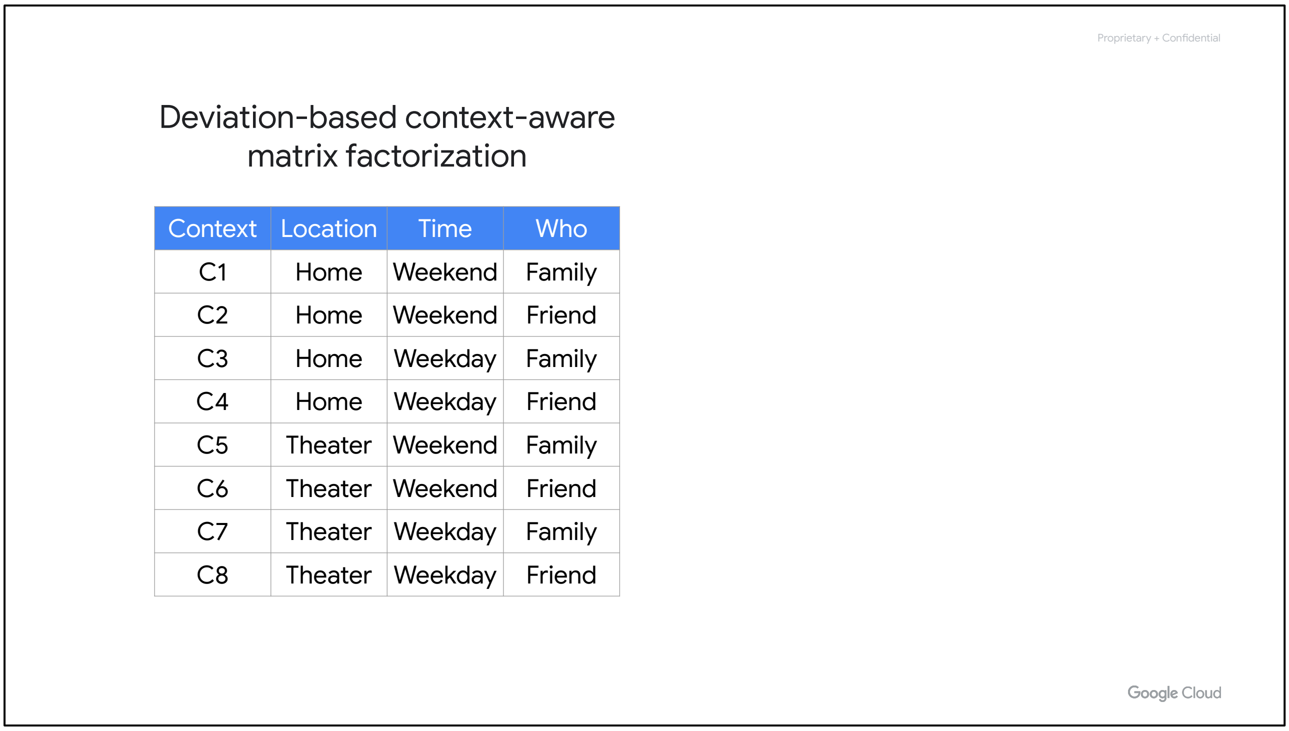
Contextual modeling has many different algorithms to learn multi-dimensional models of contextual user-item interactions. Factorization became very popular and has led to many methods over the years. Let's take a deeper look at deviation-based context-aware matrix factorization.



Deviation-based context-aware matrix factorization

- How is user's rating deviated?
- Contextual rating deviation (CRD)
- Looks at the deviations of users across context dimensions

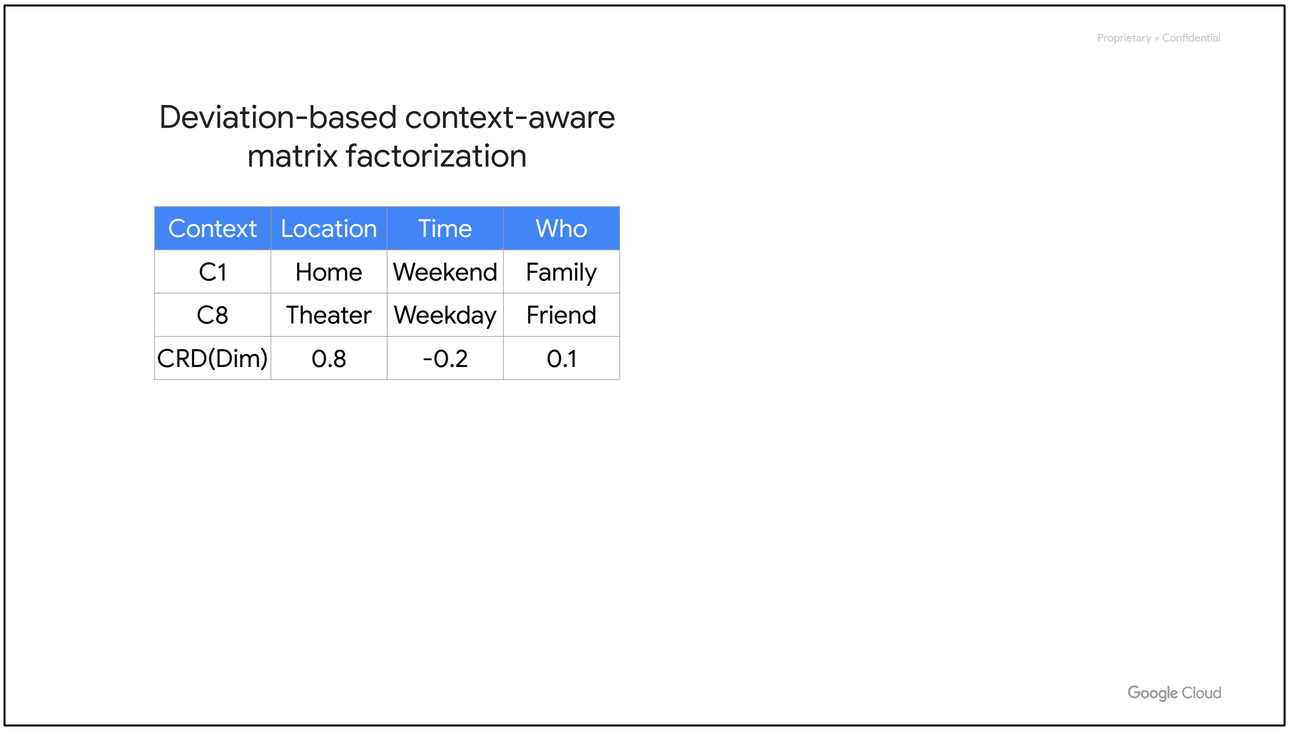
In deviation-based context-aware matrix factorization, we want to know how a user's rating is deviated across contexts. This difference is called the contextual rating deviation, or CRD. It looks at the deviations of users across context dimensions. Let's look at a quick example.



Deviation-based context-aware matrix factorization

Context	Location	Time	Who
C1	Home	Weekend	Family
C2	Home	Weekend	Friend
C3	Home	Weekday	Family
C4	Home	Weekday	Friend
C5	Theater	Weekend	Family
C6	Theater	Weekend	Friend
C7	Theater	Weekday	Family
C8	Theater	Weekday	Friend

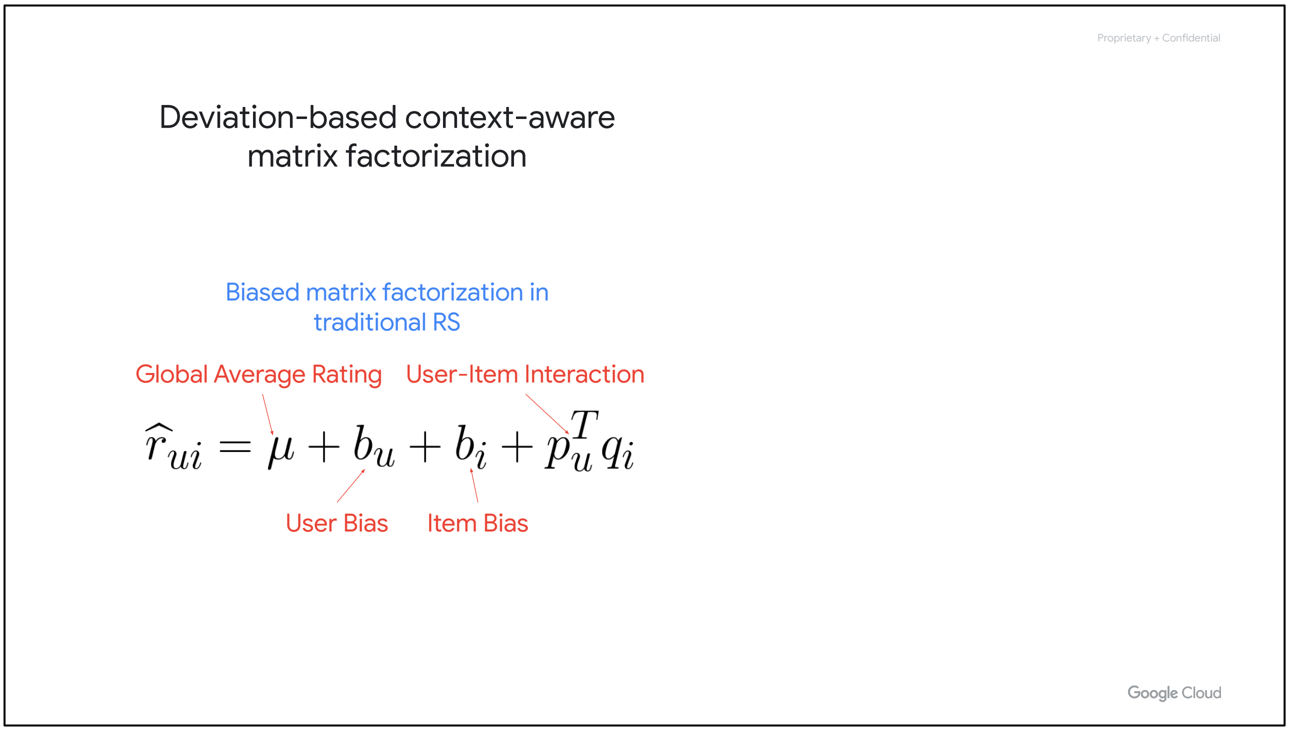
Here is an example of some of our contextual data for movies. Here we have essentially grouped by all of the unique contexts, which in this example we have 8 of, with 3 dimensions each: location, time, and who the user watched the movie with.



Deviation-based context-aware matrix factorization

Context	Location	Time	Who
C1	Home	Weekend	Family
C8	Theater	Weekday	Friend
CRD(Dim)	0.8	-0.2	0.1

Let's only look at two contexts to keep things simpler. We can calculate the contextual rating deviation, or CRD, for each of these context dimensions. The CRD for location is 0.8, which means that users' ratings in the location dimension are generally 0.8 higher for theater than home. The CRD for time is -0.2, which means that users' ratings in the time dimension are generally 0.2 lower for weekday than weekend. The CRD for who the user watched the movie with is 0.1, which is generally 0.1 higher for friend than for family. So how do we use the contextual rating deviation to adjust rating predictions?



Deviation-based context-aware matrix factorization

Biased matrix factorization in traditional RS

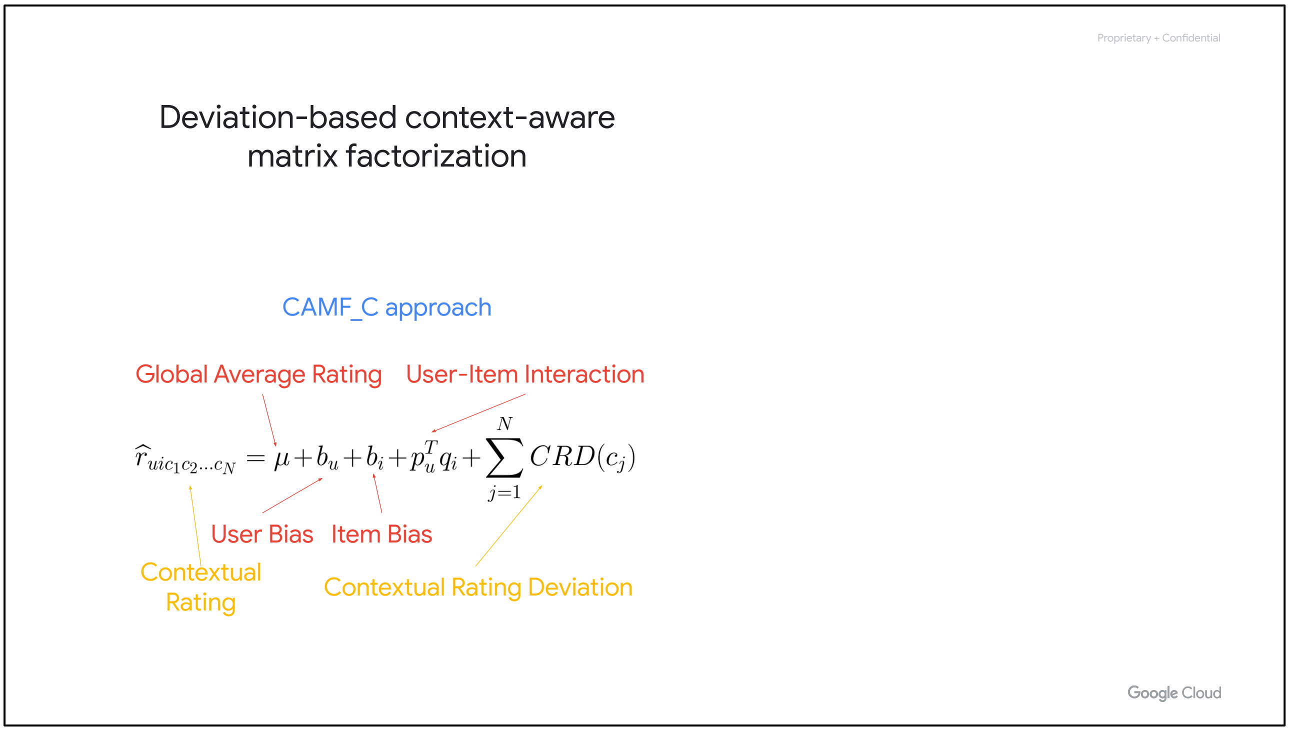
Global Average Rating User-Item Interaction

$$\widehat{r}_{ui} = \mu + b_u + b_i + p_u^T q_i$$

User Bias Item Bias

In traditional recommendation systems, when finding our ratings, we can use standard matrix factorization or we can use biased matrix factorization, where we add a term for the global average rating, a bias term for user u, and a bias term for item i. Of course, we have our user-item interaction term, which is the dot product of the user's vector, p, from the user-factor embedding matrix, U, and the item's vector, q, from the item-factor embedding matrix, V. As we can see, context is completely missing from our rating predictions.

\widehat{r}_{ui}=\mu +b_u+b_i+p_u^Tq_i



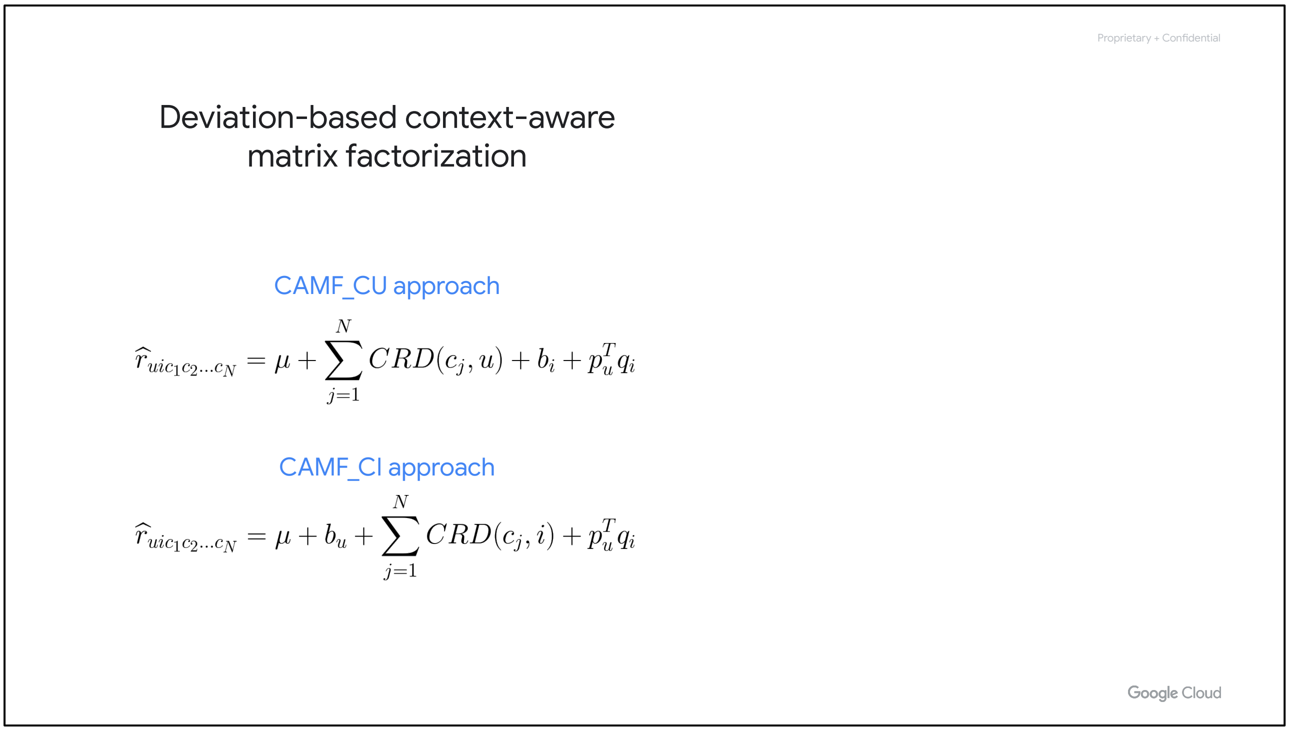
Deviation-based context-aware matrix factorization

CAMF_C approach

$$\widehat{r}_{uic_1c_2...c_N}=\mu+b_u+b_i+p_u^Tq_i+\sum_{j=1}^{N}CRD(c_j)$$

Instead of using the previous equation, we can use the context-aware matrix factorization context approach. We can see that almost everything is the same, except for two terms. On the right-hand side, we have added the contextual rating deviations summed across contexts. This gives us contextual multi-dimensional ratings on the left-hand side. Yet, this isn't the only deviation-based context-aware matrix factorization approach. Let's see some others.

\widehat{r}_{uic_1c_2...c_N}=\mu +b_u+b_i+p_u^Tq_i+\sum_{j=1}^{N}CRD(c_j)



Deviation-based context-aware matrix factorization

CAMF_CU approach

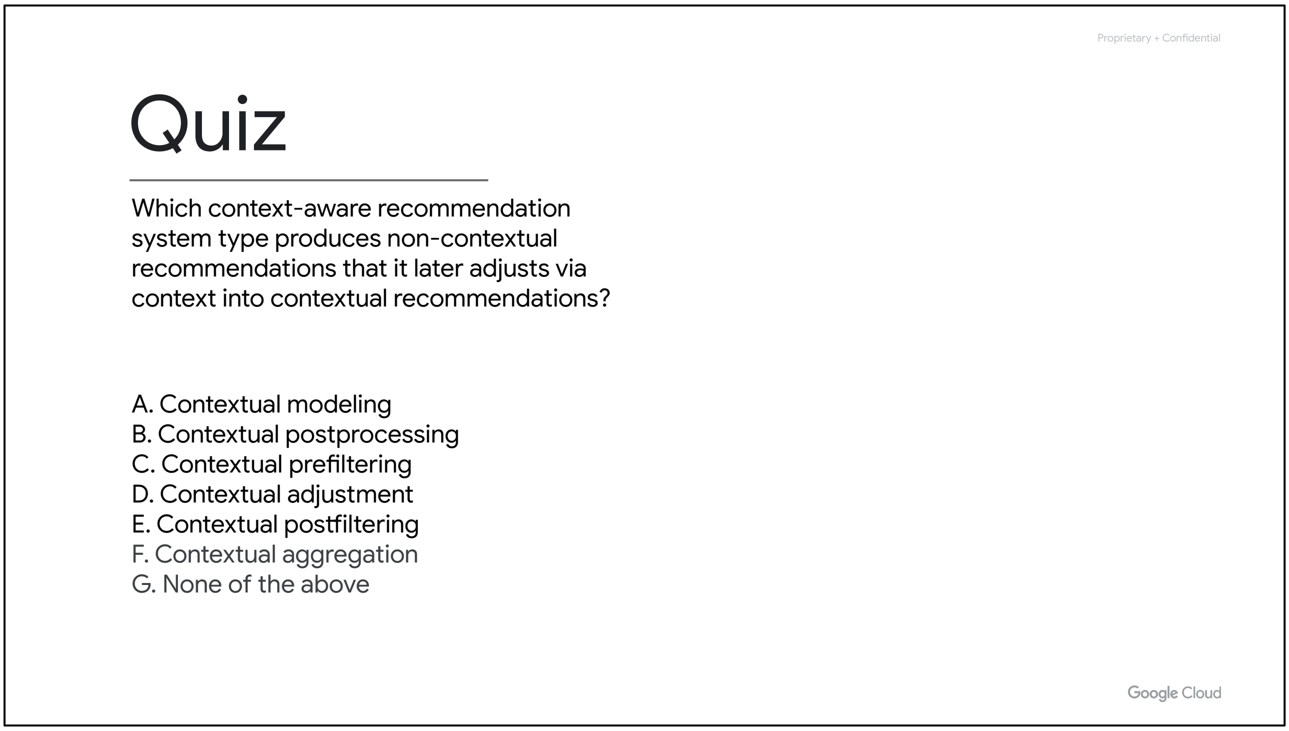
$$\widehat{r}_{uic_1c_2...c_N}=\mu+\sum_{j=1}^{N}CRD(c_j,u)+b_i+p_u^Tq_i$$

CAMF_CI approach

$$\widehat{r}_{uic_1c_2...c_N}=\mu+b_u+\sum_{j=1}^{N}CRD(c_j,i)+p_u^Tq_i$$

There are also the context user and context item approaches. In the context user approach, we've absorbed the user bias term into our CRD function, which is now dependent on both context and user. In the context item approach, we've absorbed the item bias term into our CRD function, which is now dependent on both context and item. Which approach works best out of the three? It all really depends on your problem, data, etc., so please experiment and try them all!

\widehat{r}_{uic_1c_2...c_N}=\mu +\sum_{j=1}^{N}CRD(c_j,u)+b_i+p_u^Tq_i
\widehat{r}_{uic_1c_2...c_N}=\mu +b_u+\sum_{j=1}^{N}CRD(c_j,i)+p_u^Tq_i



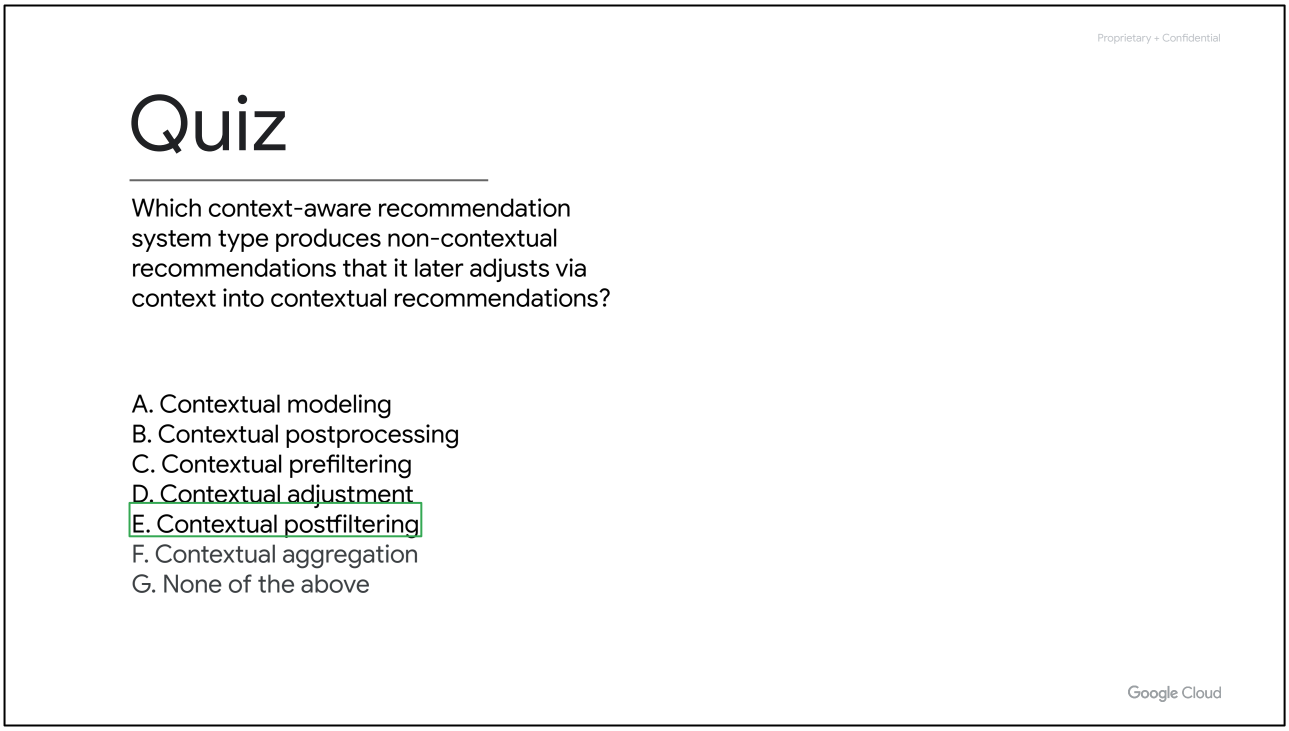
Quiz

Which context-aware recommendation system type produces non-contextual recommendations that it later adjusts via context into contextual recommendations?

A. Contextual modeling
B. Contextual postprocessing
C. Contextual prefiltering
D. Contextual adjustment
E. Contextual postfiltering
F. Contextual aggregation
G. None of the above

Google Cloud

Now that we've learned about the three main types of context-aware recommendation system algorithms, let's test your knowledge! Which context-aware recommendation system type produces non-contextual recommendations that it later adjusts via context into contextual recommendations? Contextual modeling? Contextual postprocessing? Contextual prefiltering? Contextual adjustment? Contextual postfiltering? Contextual aggregation? Or none of the above?

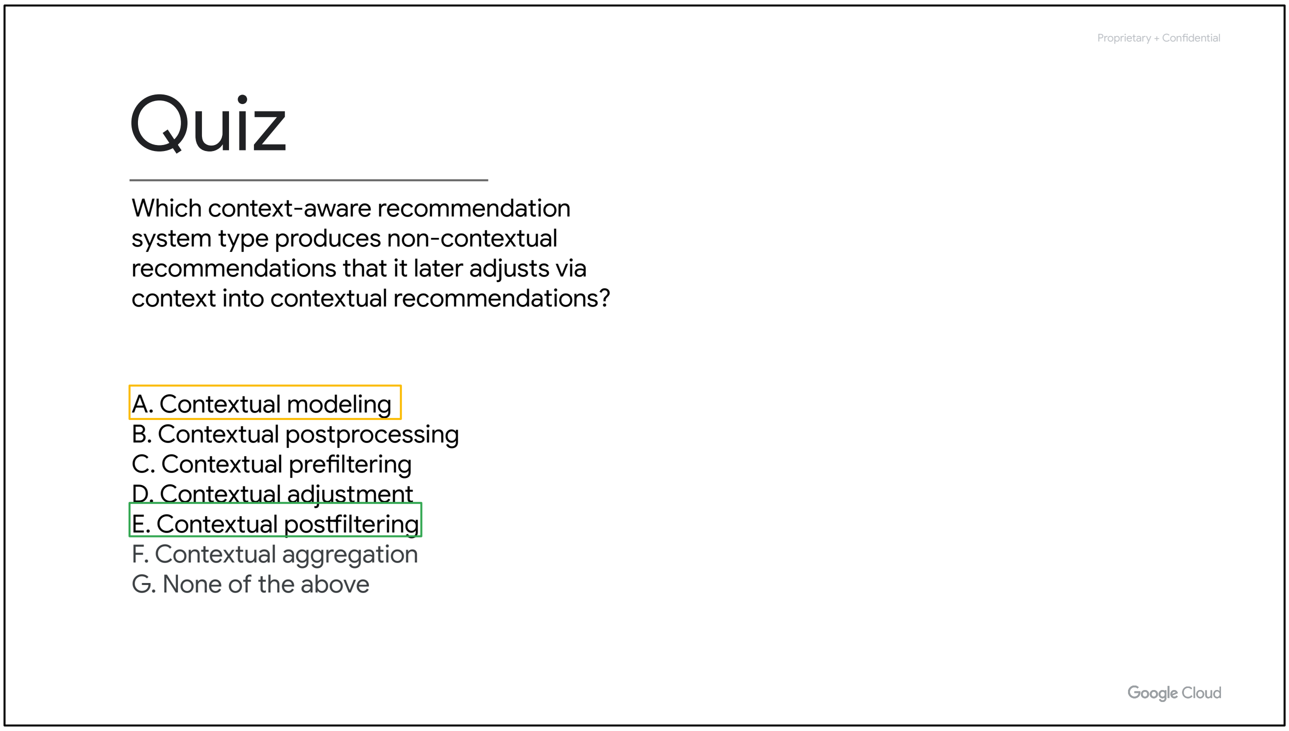


Quiz

Which context-aware recommendation system type produces non-contextual recommendations that it later adjusts via context into contextual recommendations?

A. Contextual modeling
B. Contextual postprocessing
C. Contextual prefiltering
D. Contextual adjustment
[✓ E. Contextual postfiltering]
F. Contextual aggregation
G. None of the above

The correct answer is E! Contextual postfiltering begins with our user, item, multi-dimensional context tensor. It then completely ignores context and sends the data through a traditional, two-dimensional, user-item recommendation system, which produces non-contextual recommendations. We then use the targeted context to adjust the recommendations, as we saw with the filter and weight methods, to finally arrive at contextual recommendations.

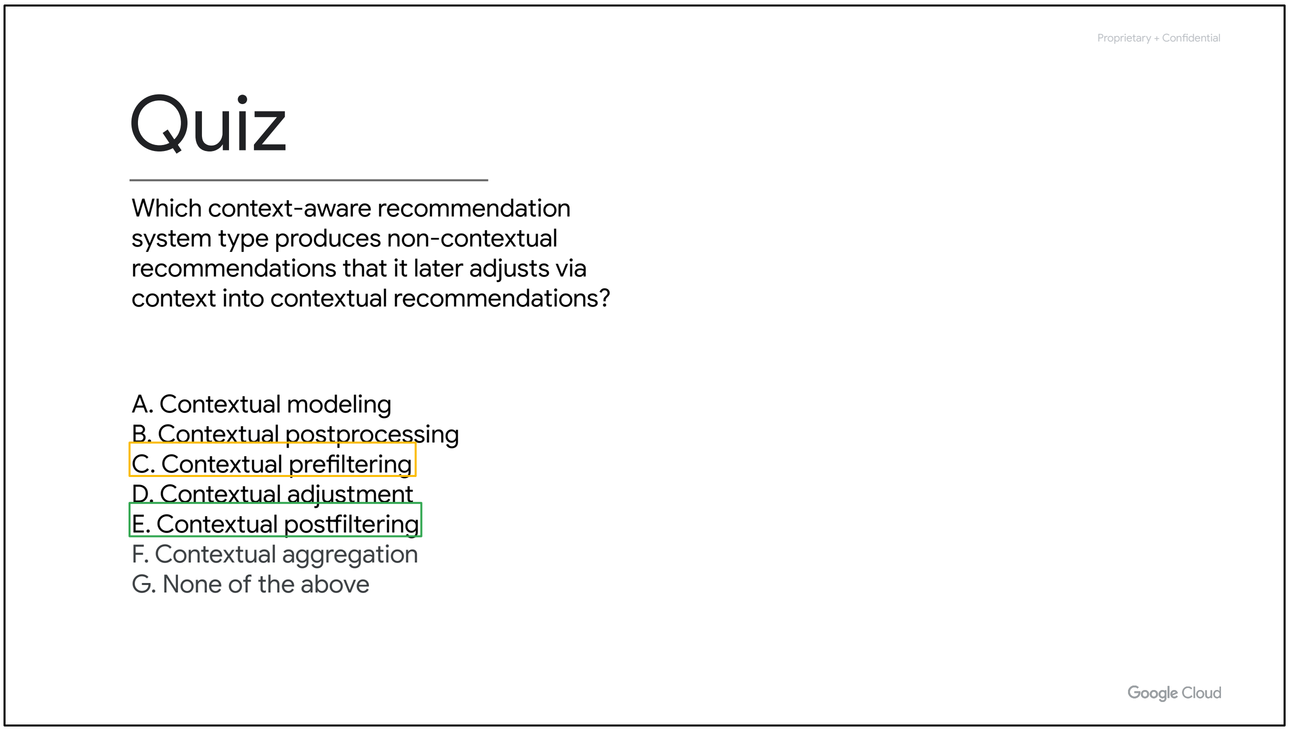


Quiz

Which context-aware recommendation system type produces non-contextual recommendations that it later adjusts via context into contextual recommendations?

[✓ A. Contextual modeling]
B. Contextual postprocessing
C. Contextual prefiltering
D. Contextual adjustment
[✓ E. Contextual postfiltering]
F. Contextual aggregation
G. None of the above

Contextual modeling actually uses an M-dimensional recommender that uses context as a explicit predictor in a prediction model, or via heuristics outputting multi-dimensional recommendations.

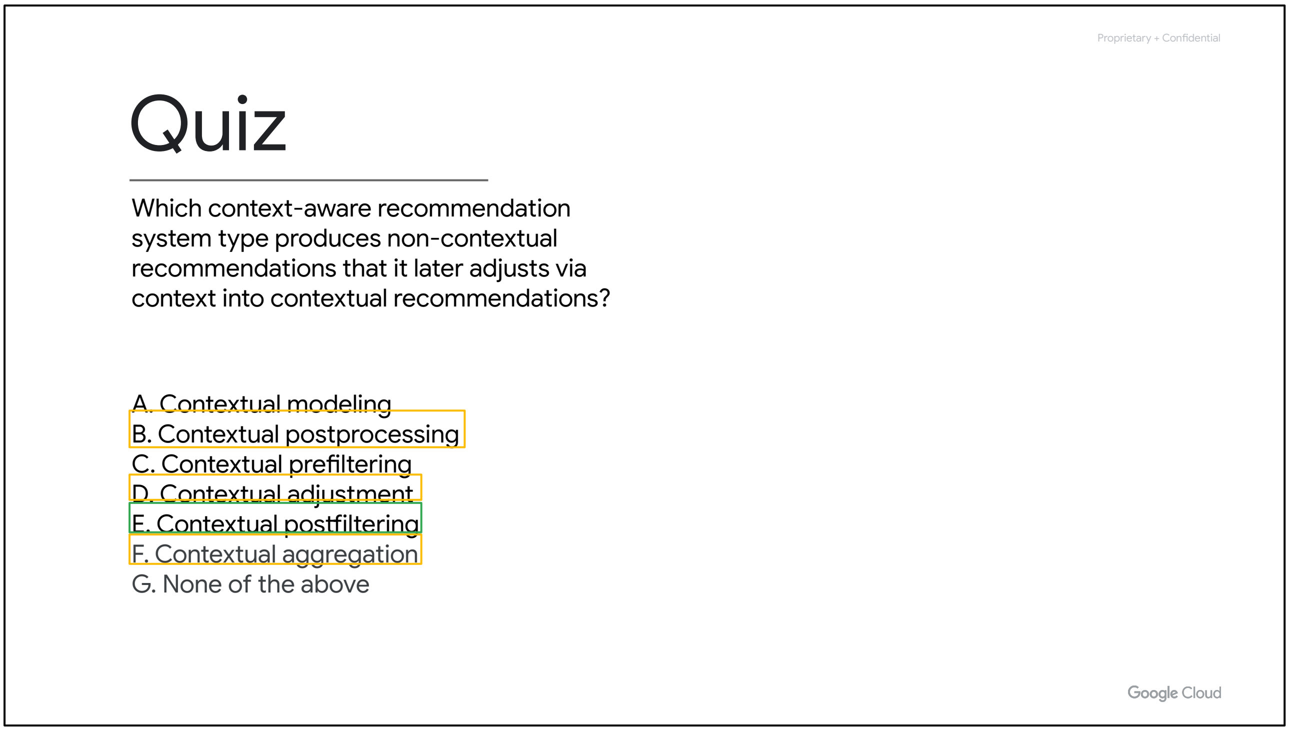


Quiz

Which context-aware recommendation system type produces non-contextual recommendations that it later adjusts via context into contextual recommendations?

A. Contextual modeling
B. Contextual postprocessing
[✓ C. Contextual prefiltering]
D. Contextual adjustment
[✓ E. Contextual postfiltering]
F. Contextual aggregation
G. None of the above

Contextual prefiltering, as the name suggests, first filters the user, item, context dataset by context, or a generalized context, to reduce the original tensor to a subset of just user-item interactions that we can use in our traditional, two-dimensional recommendation system.



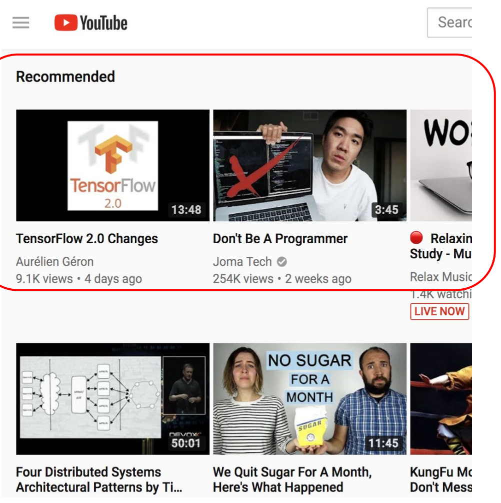
Quiz

Which context-aware recommendation system type produces non-contextual recommendations that it later adjusts via context into contextual recommendations?

A. Contextual modeling
B. Contextual postprocessing
C. Contextual prefiltering
D. Contextual adjustment
[✓ E. Contextual postfiltering]
F. Contextual aggregation
G. None of the above

The other answers we didn't cover and were just made up, but they sound cool!

YouTube



TensorFlow 2.0 Changes
Aurélien Géron
9.1K views • 4 days ago

Don't Be A Programmer
Joma Tech
254K views • 2 weeks ago

Relaxin
Study - Mu
Relax Music
1.4K watchi

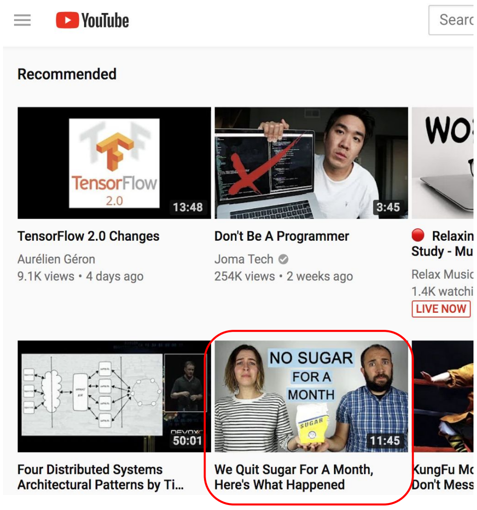
Four Distributed Systems Architectural Patterns by Ti...

We Quit Sugar For A Month, Here's What Happened

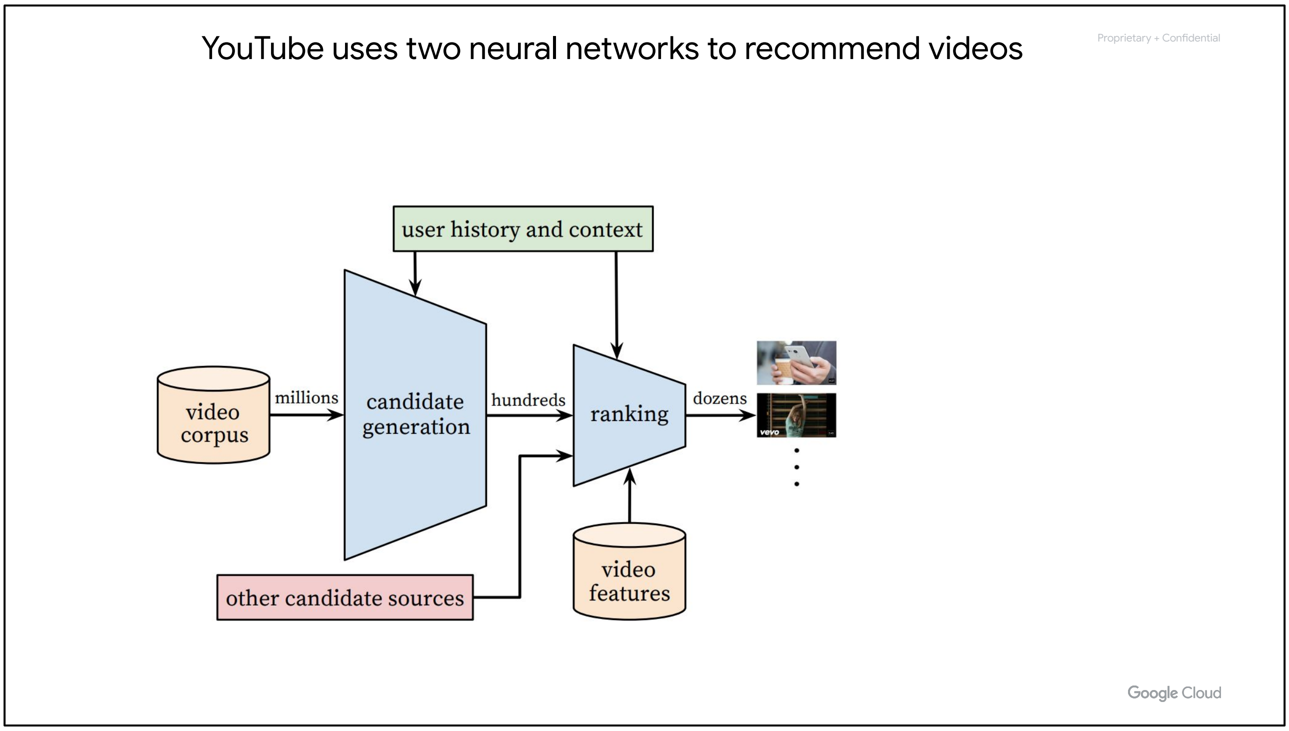
KungFu Mo
Don't Mess

Previously in this course, we saw some examples of YouTube recommendations. As we can see from the image, most recommendations are along the same vein, because perhaps this user enjoys programming, TensorFlow, and machine learning.

YouTube

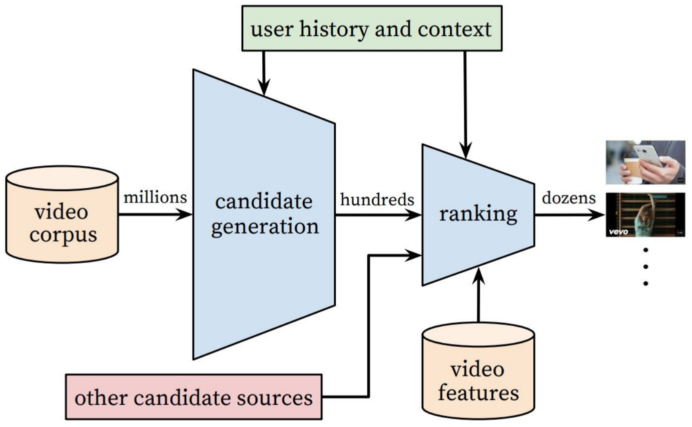


But, it also has serendipitous recommendations, such as the no sugar video, which we may like despite never thinking to search for it. There is obviously a very sophisticated hybrid recommendation system under the hood of YouTube because it typically makes great recommendations to its users. Let's now take a look underneath and see what exactly is going on.



YouTube uses two neural networks to recommend videos

Proprietary + Confidential



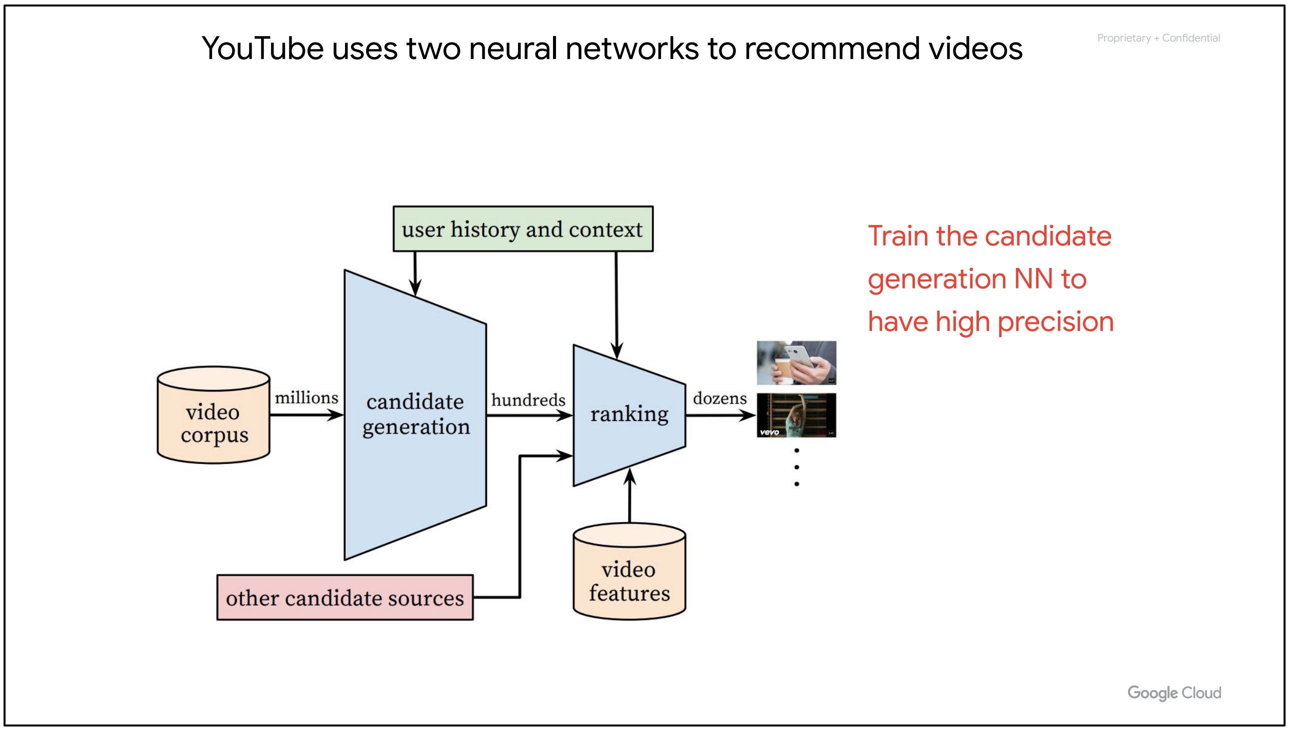
Google Cloud

YouTube uses a pretty complex hybrid recommendation system that consists of two neural networks and many different data sources to make great video recommendations.

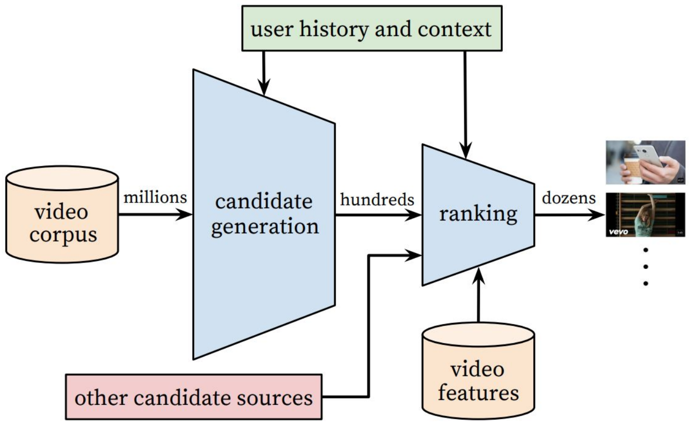
There is a candidate generation network that accepts in millions of video corpuses.

The output of this network, which is in the hundreds, is the input to a ranking network combined with other candidate sources, which could be things like videos-in-the-news, for freshness, videos for neighboring users, related videos, or sponsored videos. This is also combined with video features. This produces dozens of recommendations.

Both networks also have user history and context feeding into them to improve their recommendation quality. But training this hybrid model is not as simple as just setting an out-of-the-box loss function and letting it go.

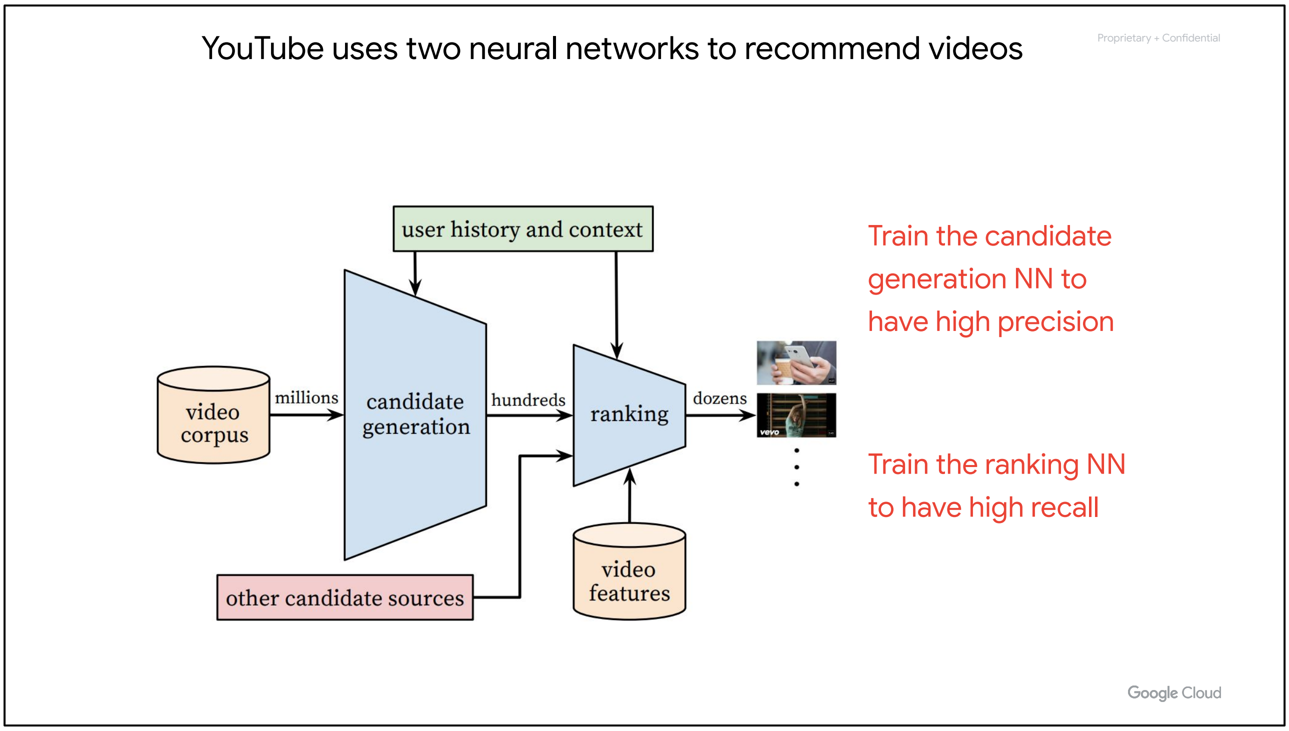


YouTube uses two neural networks to recommend videos

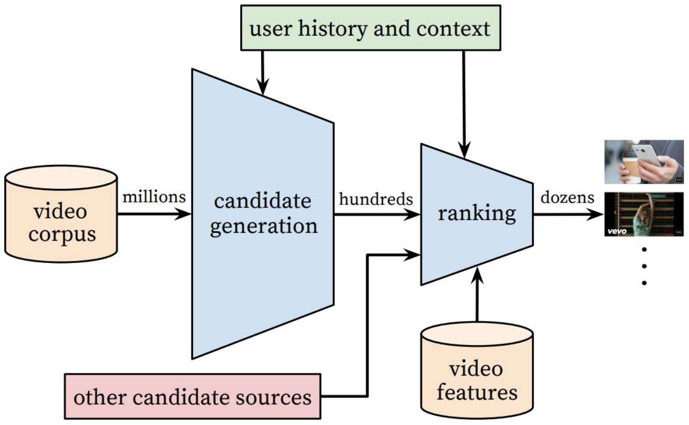


Train the candidate generation NN to have high precision

YouTube trains the candidate generation network to have high precision. High precision in this case means that every candidate generated is relevant to user. Obviously this is important, because users do not want to be shown irrelevant videos and will lose trust in the recommendations and product.



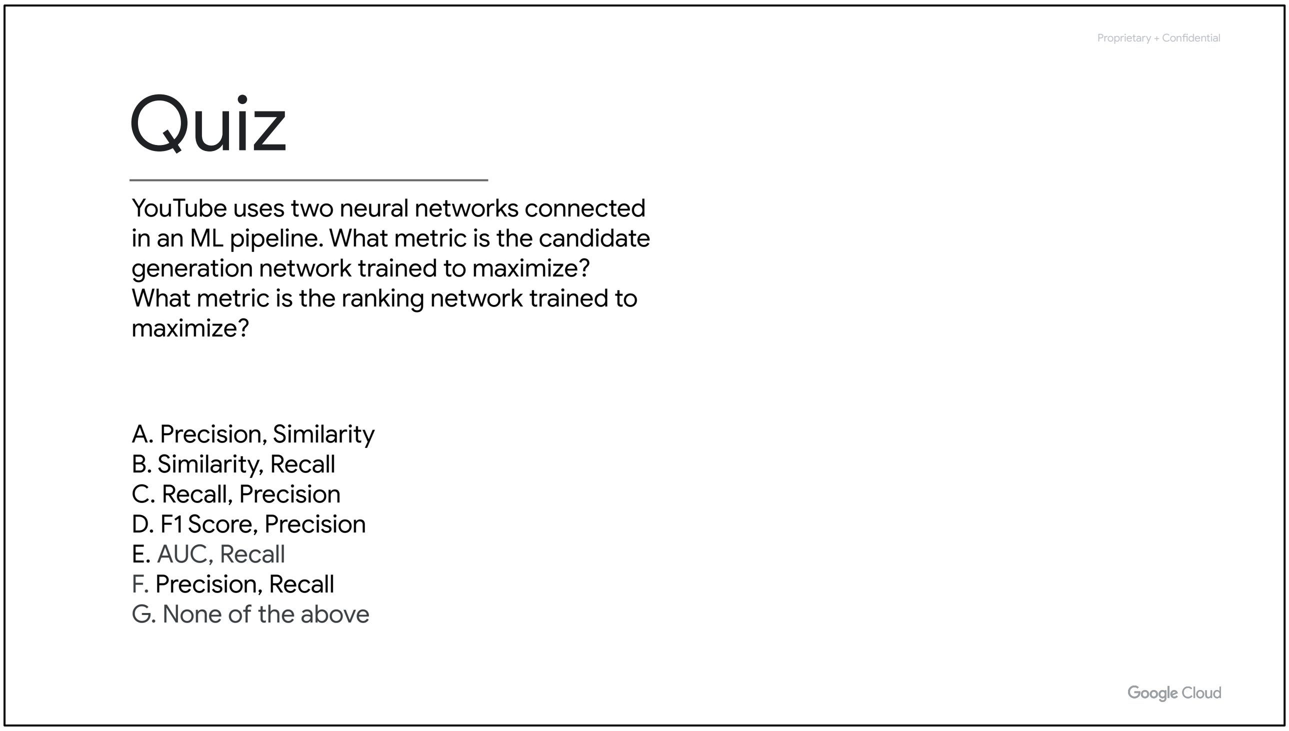
YouTube uses two neural networks to recommend videos



Train the candidate generation NN to have high precision

Train the ranking NN to have high recall

However, the second, ranking, network trains to have high recall. High recall in this case means that it will recommend things user will definitely like.

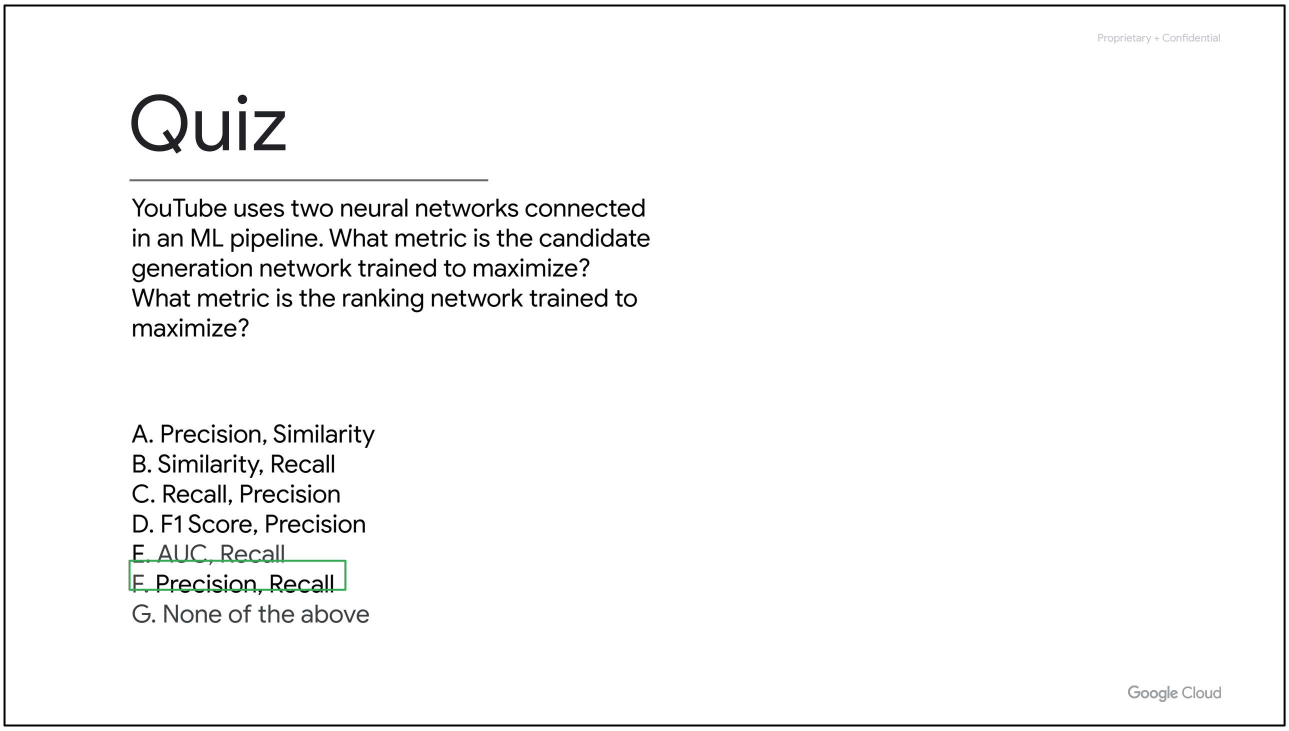


Quiz

YouTube uses two neural networks connected in an ML pipeline. What metric is the candidate generation network trained to maximize? What metric is the ranking network trained to maximize?

A. Precision, Similarity
B. Similarity, Recall
C. Recall, Precision
D. F1 Score, Precision
E. AUC, Recall
F. Precision, Recall
G. None of the above

Now that we've started to see what is under the hood of YouTube's hybrid recommendation system, let's test your knowledge! YouTube uses two neural networks connected in an ML pipeline. What metric is the candidate generation network trained to maximize? What metric is the ranking network trained to maximize? When thinking about which answer is correct, it may help to think about what each network is supposed to do, and what metric would help it do that.

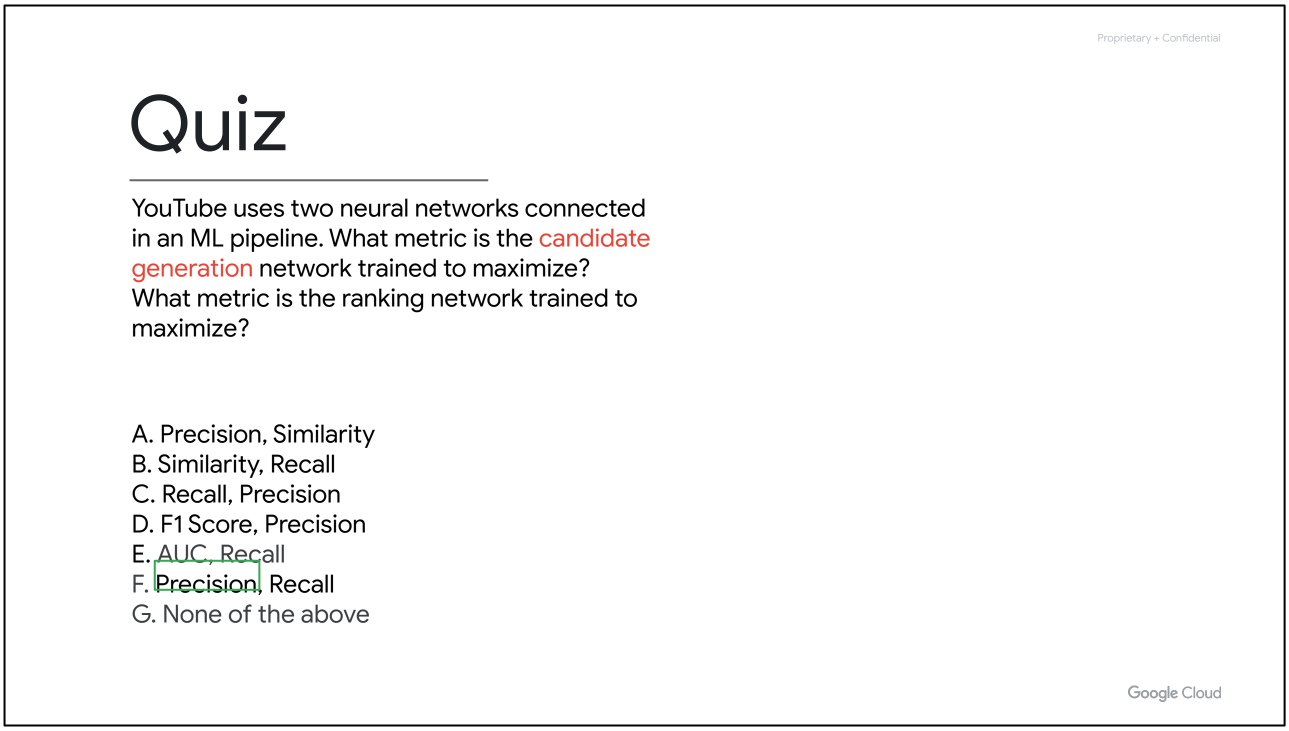


Quiz

YouTube uses two neural networks connected in an ML pipeline. What metric is the candidate generation network trained to maximize? What metric is the ranking network trained to maximize?

A. Precision, Similarity
B. Similarity, Recall
C. Recall, Precision
D. F1 Score, Precision
E. AUC, Recall
[✓ F. Precision, Recall]
G. None of the above

The correct answer is F! Precision and recall!


Proprietary + Confidential

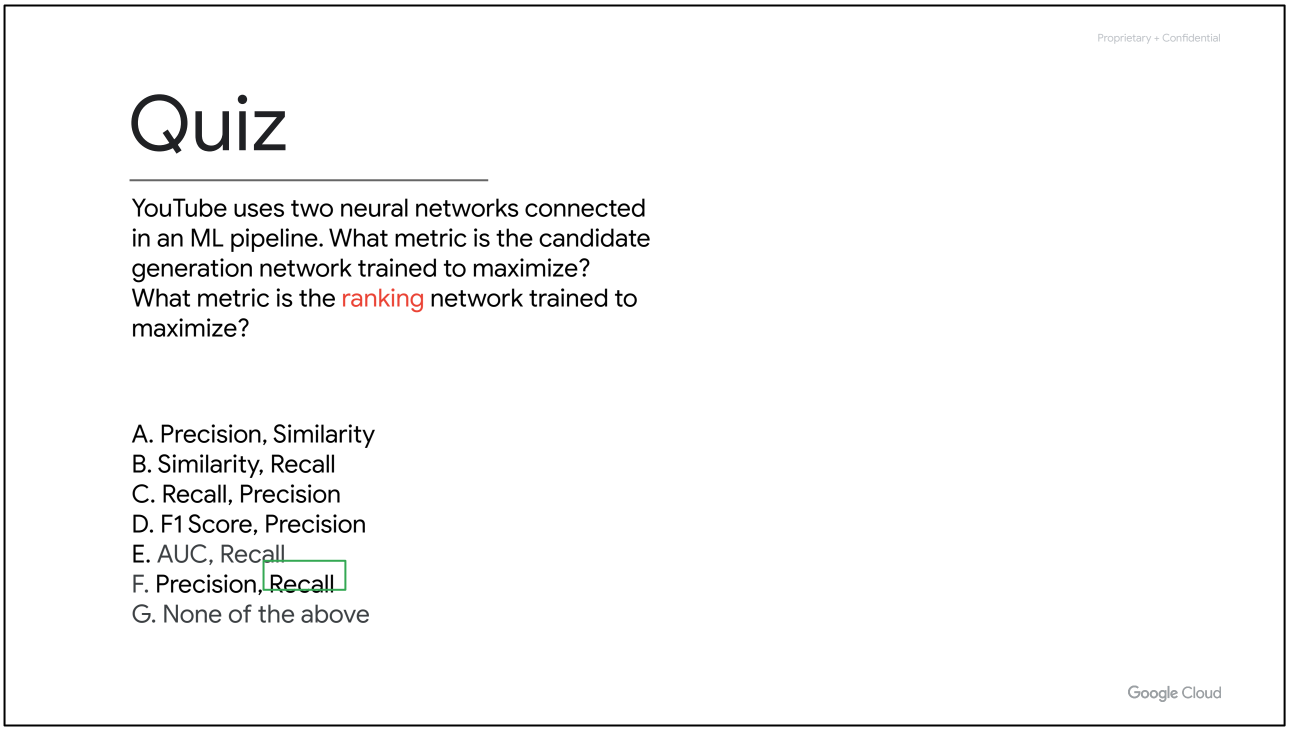
Quiz

YouTube uses two neural networks connected in an ML pipeline. What metric is the candidate generation network trained to maximize? What metric is the ranking network trained to maximize?

A. Precision, Similarity
B. Similarity, Recall
C. Recall, Precision
D. F1 Score, Precision
E. AUC, Recall
F. [✓ Precision], Recall
G. None of the above

Google Cloud

The candidate generation network is taking millions of video corpuses, as well as user and context history, to narrow down the field to hundreds of candidate videos. It wants to make sure that what it is predicting will be highly relevant to the user; therefore, to ensure that, precision is a good choice to make sure that when we say something is relevant, we can do it with high confidence.



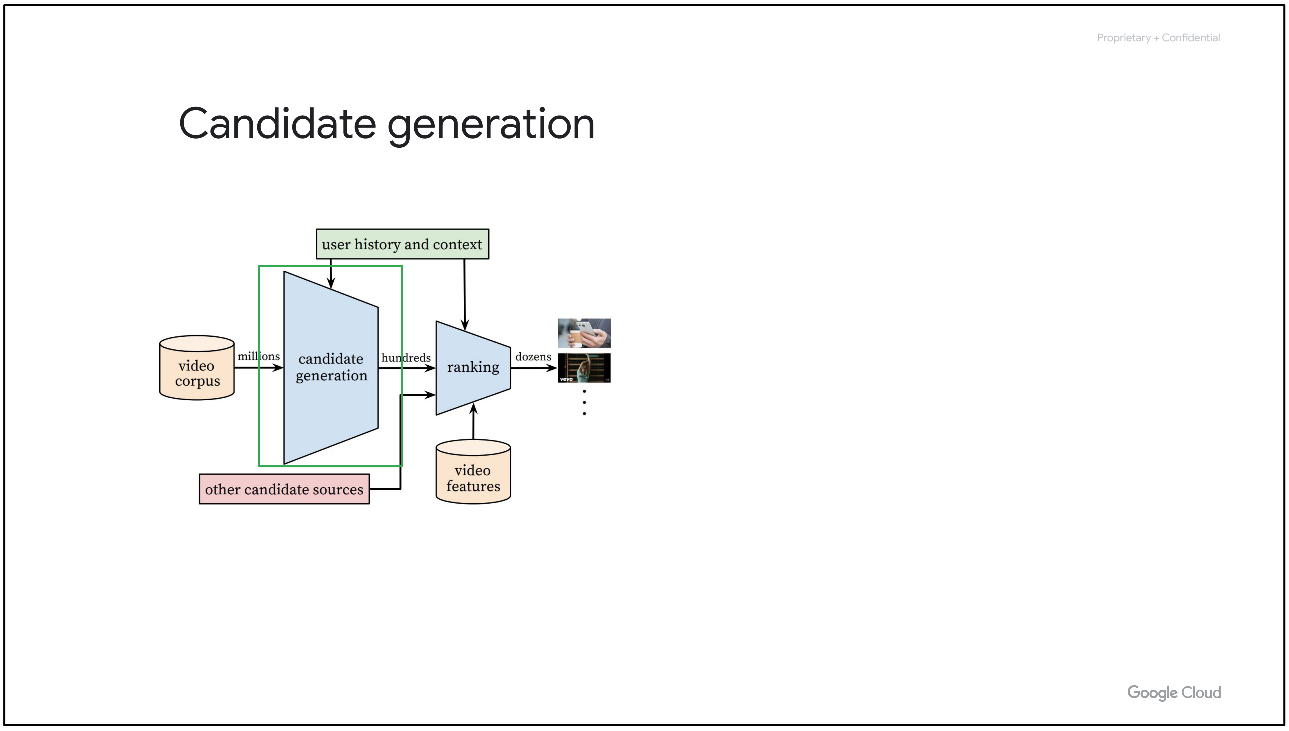
Quiz

YouTube uses two neural networks connected in an ML pipeline. What metric is the candidate generation network trained to maximize? What metric is the ranking network trained to maximize?

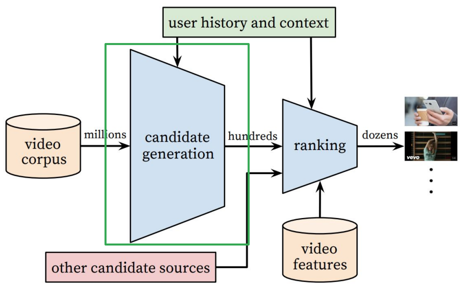
A. Precision, Similarity
B. Similarity, Recall
C. Recall, Precision
D. F1 Score, Precision
E. AUC, Recall
F. Precision, [✓ Recall]
G. None of the above

Google Cloud

The ranking network is taking these hundreds of candidate videos, combining that with video features and other candidate sources, as well as user and context history. The function of the ranking network is to narrow down the candidates even further, to only dozens, while making sure that the videos being returned will definitely be something that users like. This means that maximizing recall for this network will help make sure this happens. Therefore, users end up with relevant videos that they really like.



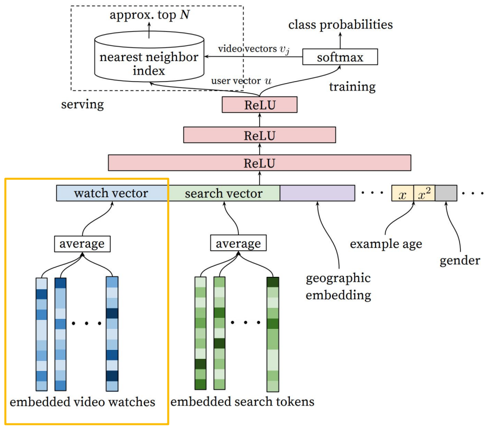
Candidate generation



Now that we've had the overview, let's take a deeper dive into each of the two neural networks used for the YouTube video recommendation system, starting with the candidate generation network.

Candidate generation consists of intelligently assembling many other ML models (1/3)

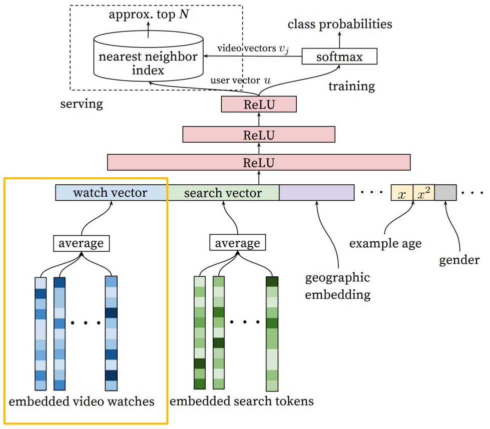
1. Get item embedding from, e.g., WALS



Just as we did in our design lab and neural network lab, hybrid models require intelligently assembling many other machine learning models into a pipeline. These use multiple datasets and model types to help give the best recommendations. The first step for candidate generation is to get the item embeddings, for instance from a trained WALS model.

Candidate generation consists of intelligently assembling many other ML models (1/3)

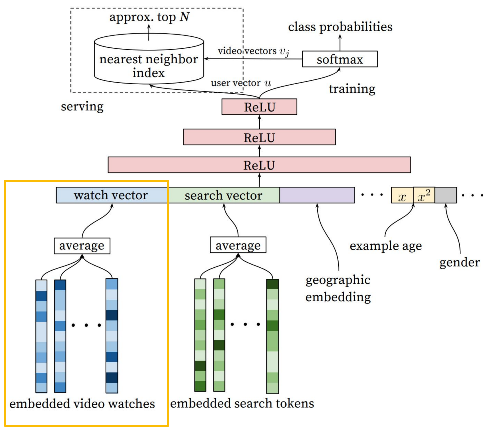
1. Get item embedding from, e.g., WALS
2. Find last 10 videos watched by user



We then find the last 10 videos watched by the user and use the embeddings to get their vectors within embedding space.

Candidate generation consists of intelligently assembling many other ML models (1/3)

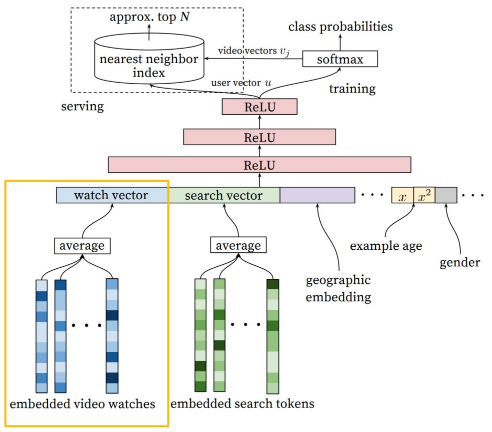
1. Get item embedding from, e.g., WALS
2. Find last 10 videos watched by user
3. Average the embeddings of those 10 videos



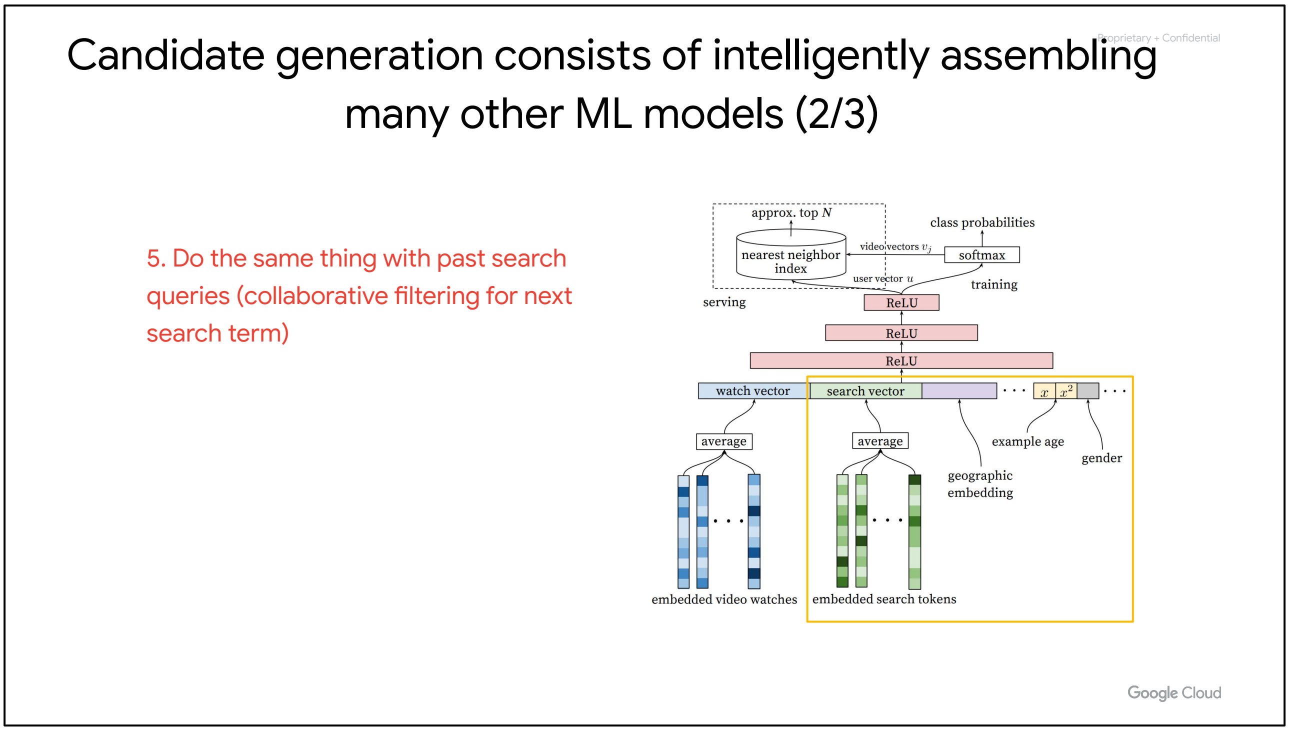
Next, we average the embeddings of those 10 videos so we will have a resultant single embedding that is the average along each embedding dimension.

Candidate generation consists of intelligently assembling many other ML models (1/3)

1. Get item embedding from, e.g., WALS
2. Find last 10 videos watched by user
3. Average the embeddings of those 10 videos
4. This is the watch vector

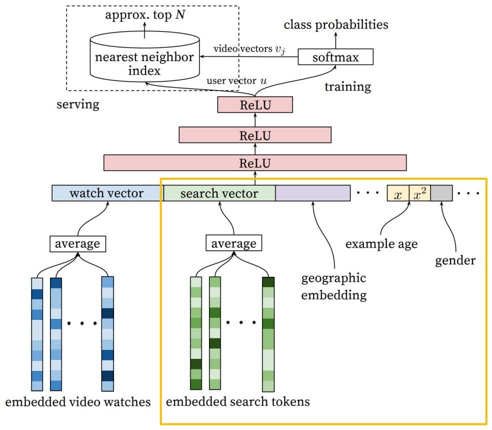


This becomes the watch vector, which will be one of the features to our deep neural network.

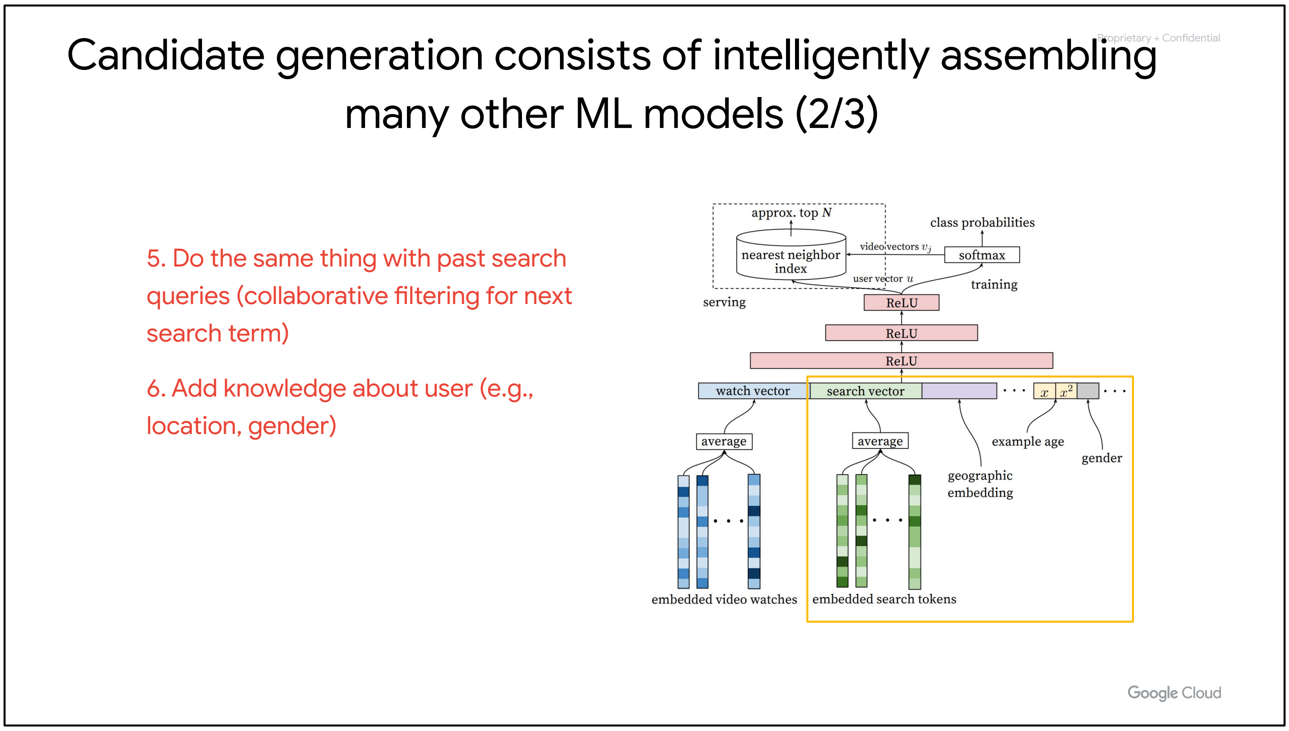


Candidate generation consists of intelligently assembling many other ML models (2/3)

5. Do the same thing with past search queries (collaborative filtering for next search term)



We will do the same thing with past search queries. Collaborative filtering for next search term is a collaborative filtering model that is similar to the user-history based collaborative filtering we talked about earlier. Essentially, this is like doing a word2vec on pairs of search terms. We will find an average search embedding, and this will become our search vector, another input feature to our deep neural network.



Candidate generation consists of intelligently assembling many other ML models (2/3)

5. Do the same thing with past search queries (collaborative filtering for next search term)

6. Add knowledge about user (e.g., location, gender)

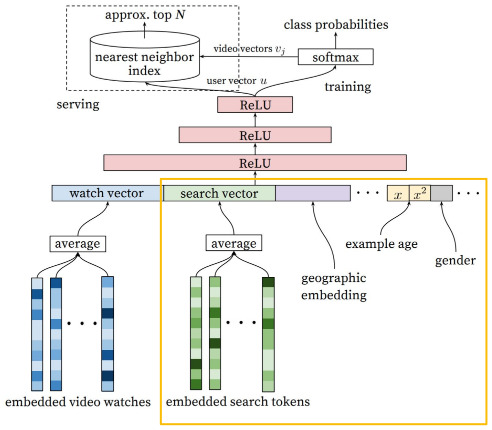
We also should add any knowledge we have about the user. Location is important so users can see localized videos and also because of language. Gender may be important because perhaps there are differences in preference based on that. All of these are features added as inputs to our deep neural network.

Candidate generation consists of intelligently assembling many other ML models (2/3)

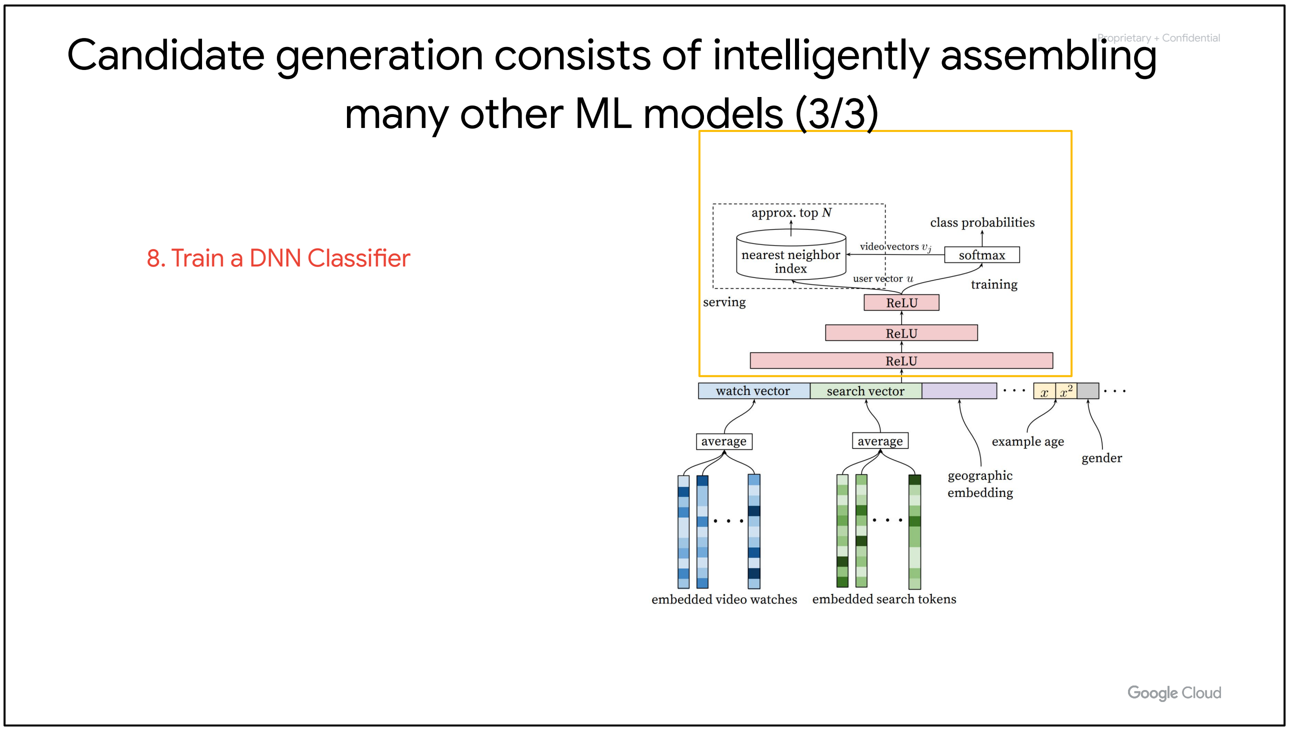
5. Do the same thing with past search queries (collaborative filtering for next search term)

6. Add knowledge about user (e.g., location, gender)

7. Add example age to avoid overemphasizing older videos

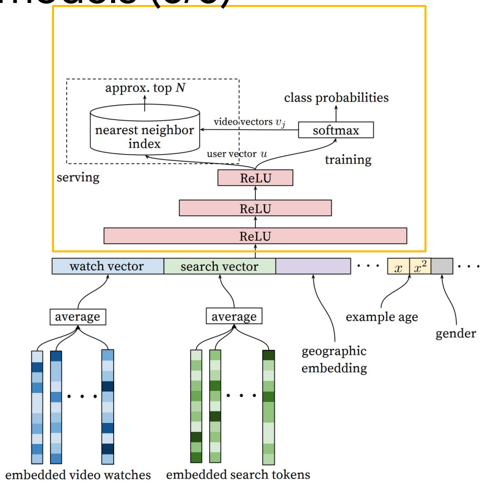


We also should add example age, because we don't want to overemphasize older videos. Why do we care about this? Well, older videos have more likes and more user interactions in general. You want the model to learn to account for the fact that older videos are more likely to have been watched or searched for. Hence, the need to add “example age” as an input. Remember, this is example age, not user age! These all are a subset of the many features that feed into our candidate generation neural network.



Candidate generation consists of intelligently assembling many other ML models (3/3)

8. Train a DNN Classifier



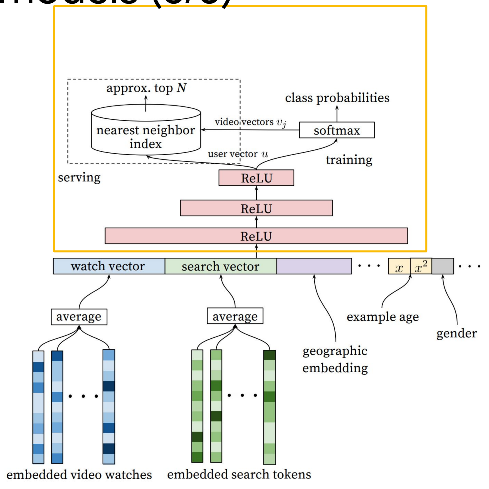
Google Cloud

Next, we want to train a DNN Classifier. Now why would we want this model type? When you think of a DNN Classifier's output it becomes pretty obvious. The output will be the probability that this video will be watched. So, taking the top-N of these videos is probably what we want. These come out of the softmax for training.

Candidate generation consists of intelligently assembling many other ML models (3/3)

8. Train a DNN Classifier

9. Treat the last-but-one layer as a user embedding



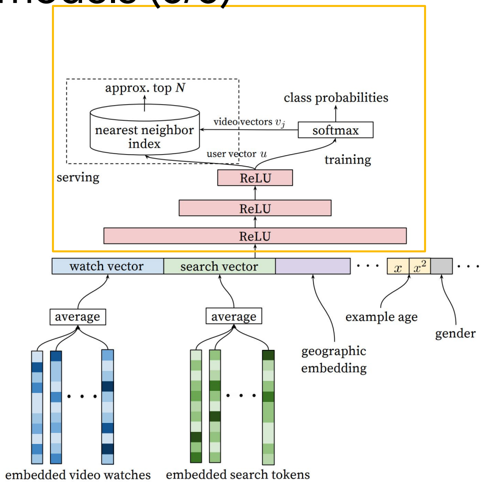
But there are two other considerations:
There is also a benefit to finding the closest users, and generating those candidates as well. This is the way that viral videos are created, videos that a lot of people like us are watching. Therefore, we can treat the last-but-one layer, or the layer right before the softmax, as a user embedding.

Candidate generation consists of intelligently assembling many other ML models (3/3)

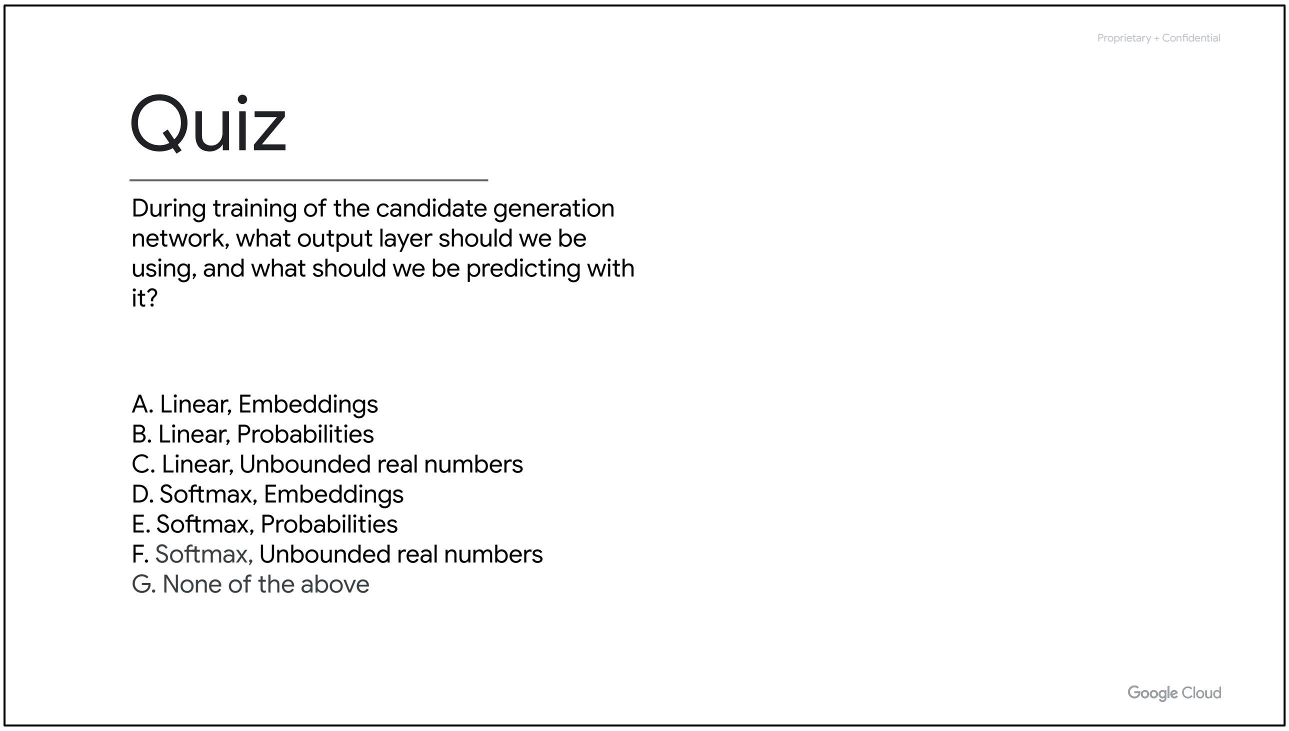
8. Train a DNN Classifier

9. Treat the last-but-one layer as a user embedding

10. Use output of DNN Classifier and user embedding to generate candidates



There is also a benefit to finding videos related, content-wise, to the video you are currently watching. Therefore, we can use the output of the DNN Classifier as video vectors. This combined with the last ReLU layer, as the user embeddings, generates candidates during serving. So nearest neighbors consists of neighboring users and neighboring videos.

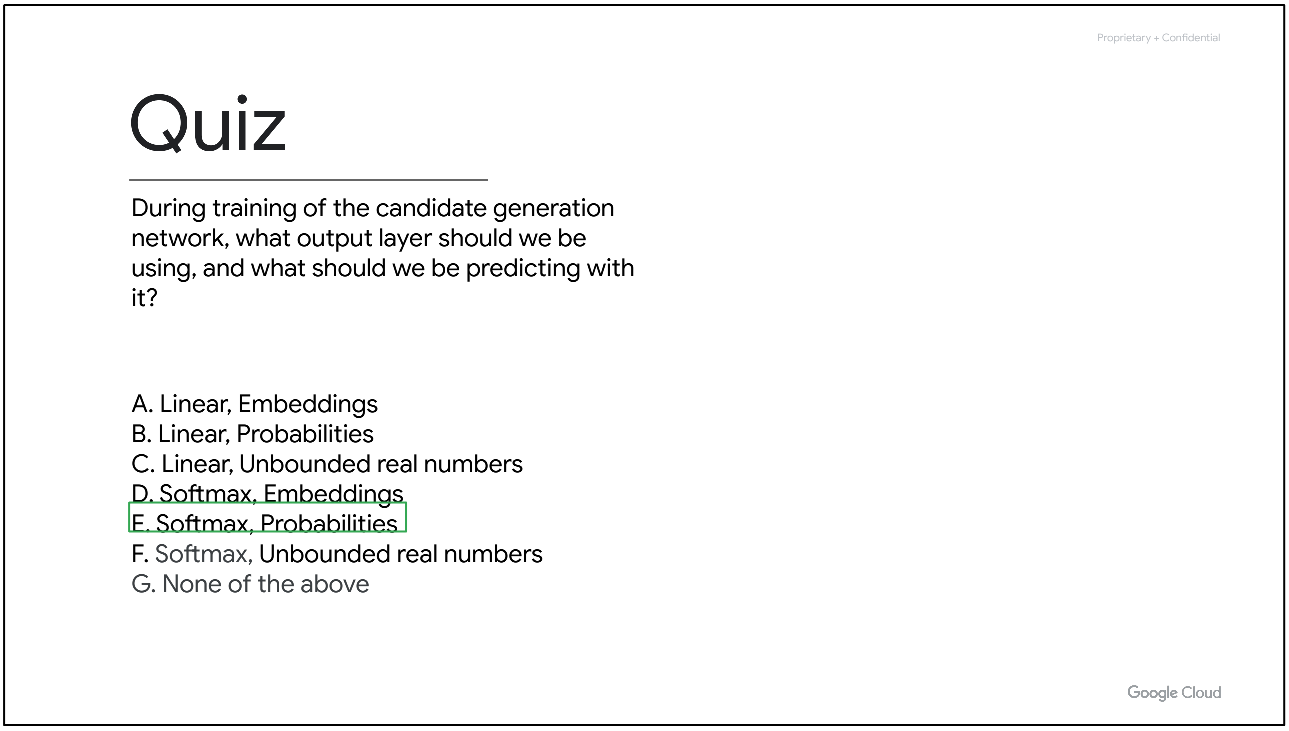


Quiz

During training of the candidate generation network, what output layer should we be using, and what should we be predicting with it?

A. Linear, Embeddings
B. Linear, Probabilities
C. Linear, Unbounded real numbers
D. Softmax, Embeddings
E. Softmax, Probabilities
F. Softmax, Unbounded real numbers
G. None of the above

We now know the magic behind YouTube's candidate generation neural network, so let's see what we've learned. During training of the candidate generation network, what output layer should we be using and what should we be predicting with it?

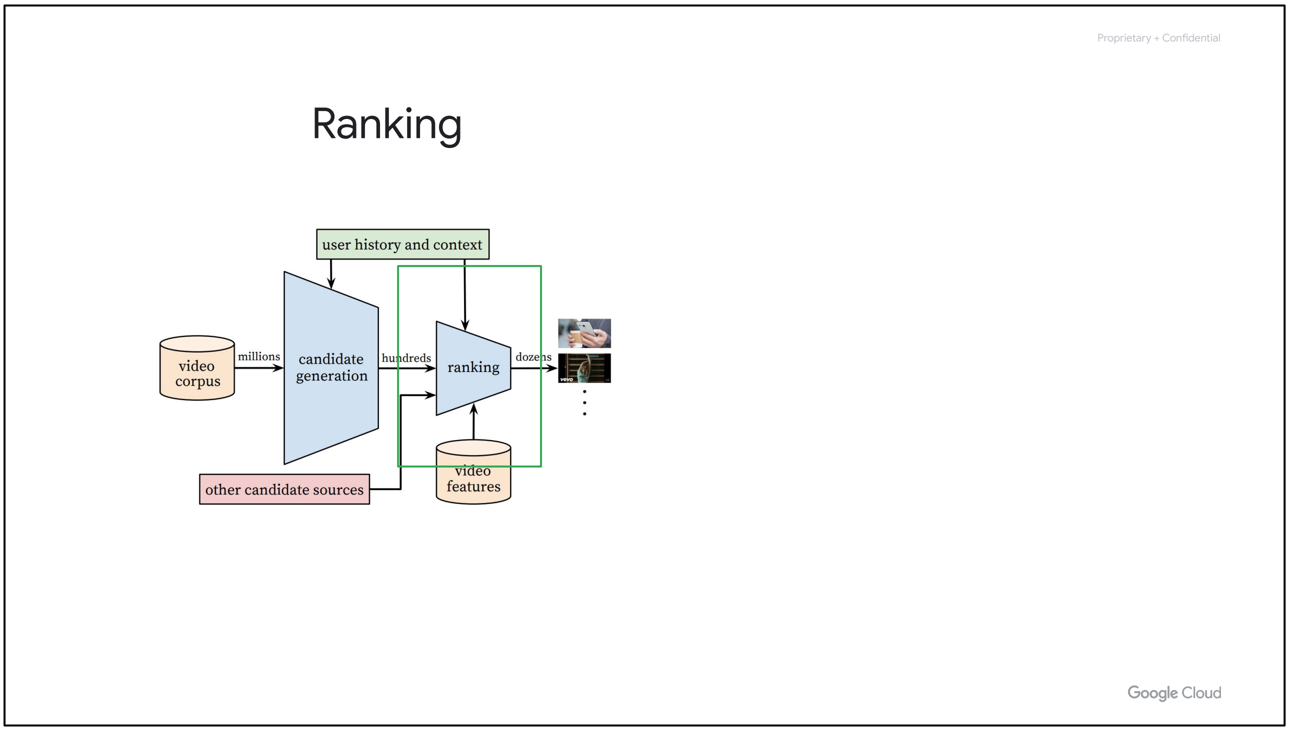


Quiz

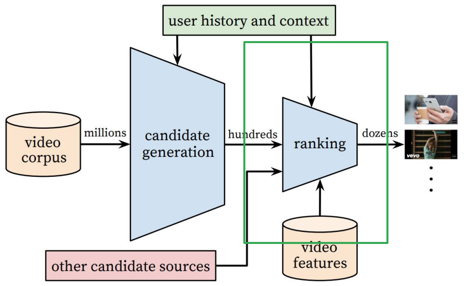
During training of the candidate generation network, what output layer should we be using, and what should we be predicting with it?

A. Linear, Embeddings
B. Linear, Probabilities
C. Linear, Unbounded real numbers
D. Softmax, Embeddings
[✓ E. Softmax, Probabilities]
F. Softmax, Unbounded real numbers
G. None of the above

The correct answer is E! We are trying to generate candidate videos to send on to the ranking neural network half of our hybrid recommendation system. Therefore, by using a softmax, we can get probabilities of which video will be watched, and we can then take the highest ones and send those as our candidate videos to the ranking network.



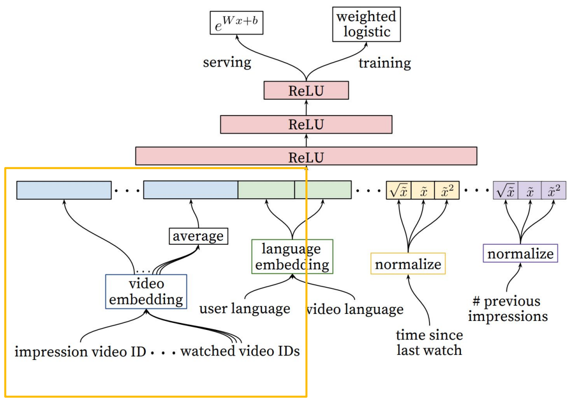
Ranking



Great! We've narrowed down our original huge dataset of videos to only hundreds of candidates. Now let's deep dive into the ranking neural network.

The ranking network uses more tailored features (1/3)

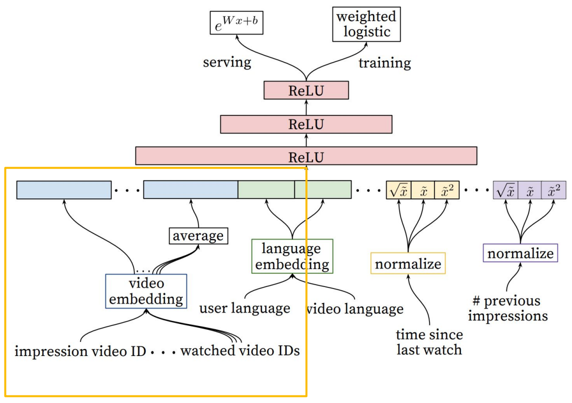
1. Videos suggested to user



Now that we have our hundreds of candidate videos that have been optimized for high precision to ensure high relevance, the ranking network uses more tailored features to make sure the user will like the videos. The first step is to take the videos suggested to the user.

The ranking network uses more tailored features (1/3)

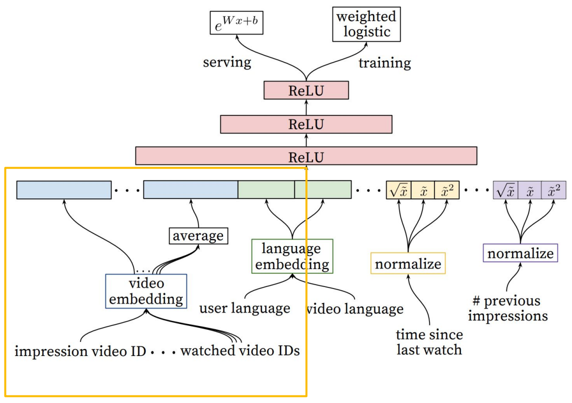
1. Videos suggested to user
2. Videos watched by user



These get combined with the videos watched by the user.

The ranking network uses more tailored features (1/3)

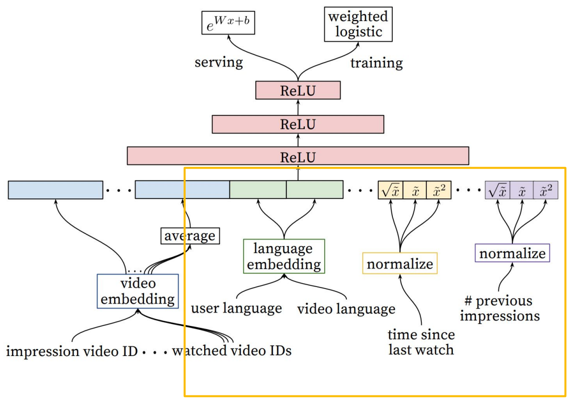
1. Videos suggested to user
2. Videos watched by user
3. Both individual and average embedding



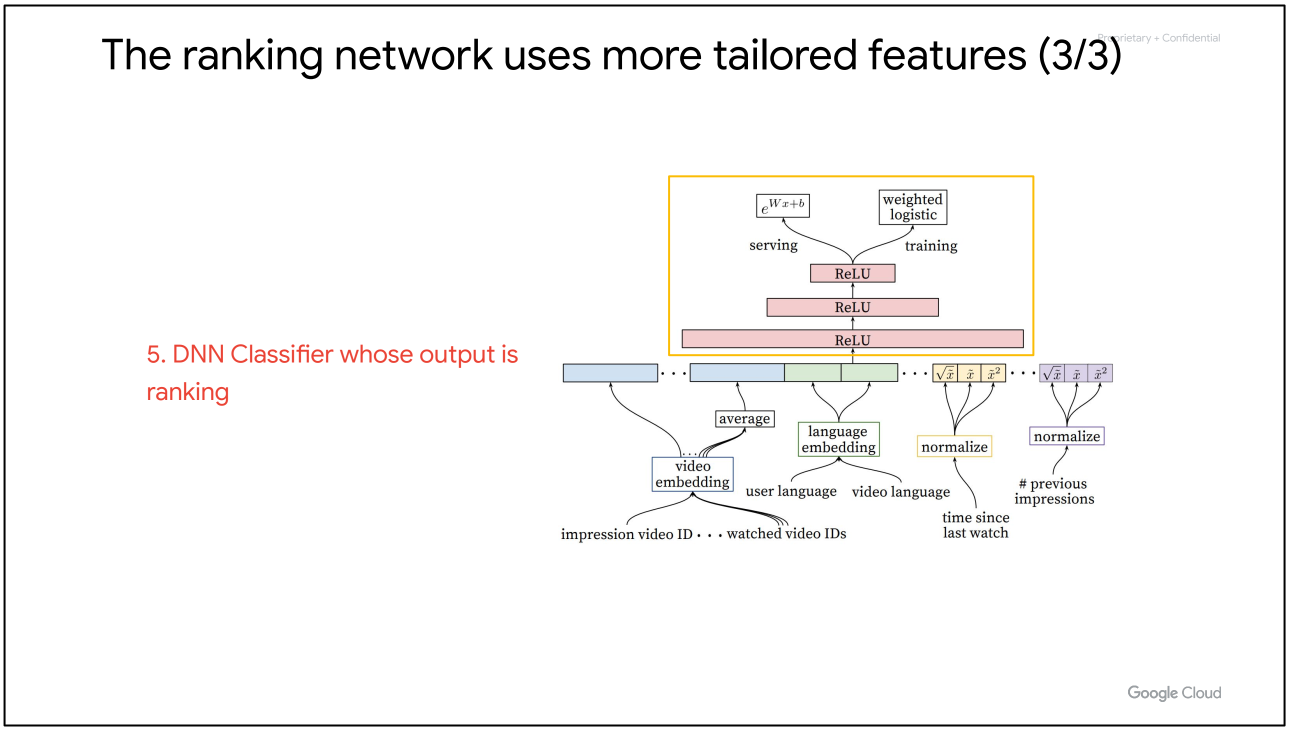
These both get used as individual video embeddings and as an average embedding, both of which are used as input features to the ranking neural network.

The ranking network uses more tailored features (2/3)

4. Hundreds of features including language embeddings and user behavior

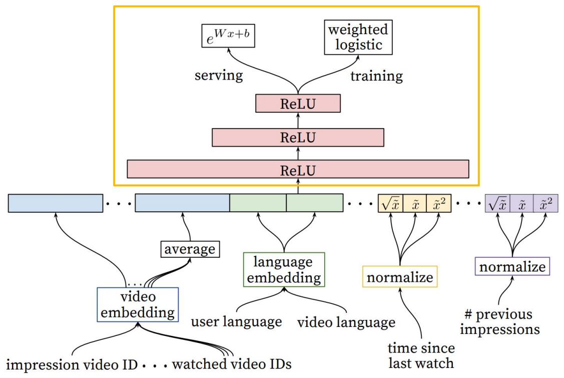


We then add hundreds of features, including language embeddings and user behavior. Why use a language embedding of both user language and video language? The reason to use this language embedding is to consider language pairs. For instance, Germans tend to watch English videos, but not vice versa. By having this embedding, we are able to capture this difference.



The ranking network uses more tailored features (3/3)

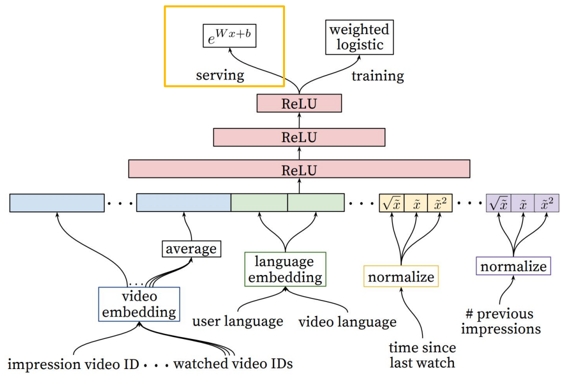
5. DNN Classifier whose output is ranking



These input features all feed through the ranking neural network's layers. The output of the DNN classifier is the ranking. During training time, the output is a weighted logistic function, whereas during serving, it is just a logistic.

The ranking network uses more tailored features (3/3)

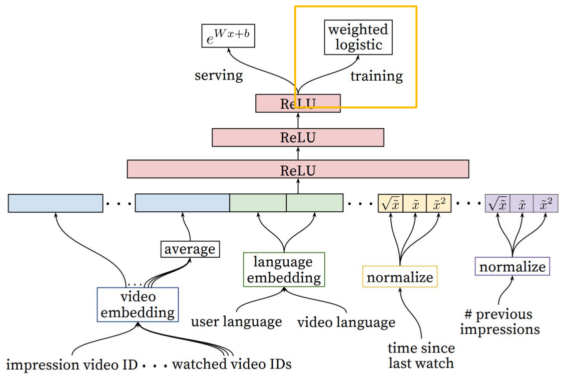
5. DNN Classifier whose output is ranking



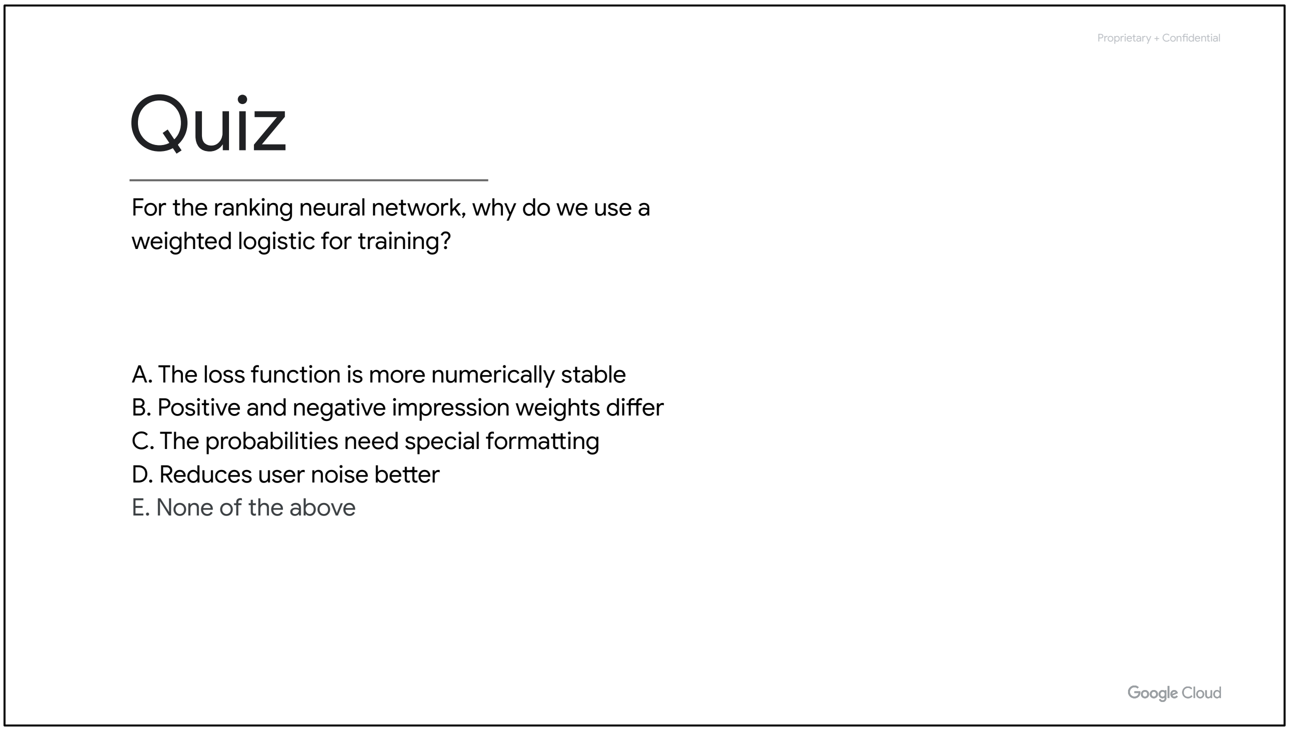
Serving uses logistic regression for scoring the videos and is optimized for expected watch time from the training labels. This is used instead of expected click, because then the recommendations could favor clickbait videos over actual good recommendations. Remember, it is important to not only use the right model architecture, but also use the right data, which includes the labels.

The ranking network uses more tailored features (3/3)

5. DNN Classifier whose output is ranking



For training, because we are using expected watch time, we use the weighted logistic instead. The watch time is used as the weight for positive interactions, and negative interactions just get a unit weight. Because the number of positive impressions compared to the total is small, this works pretty well.

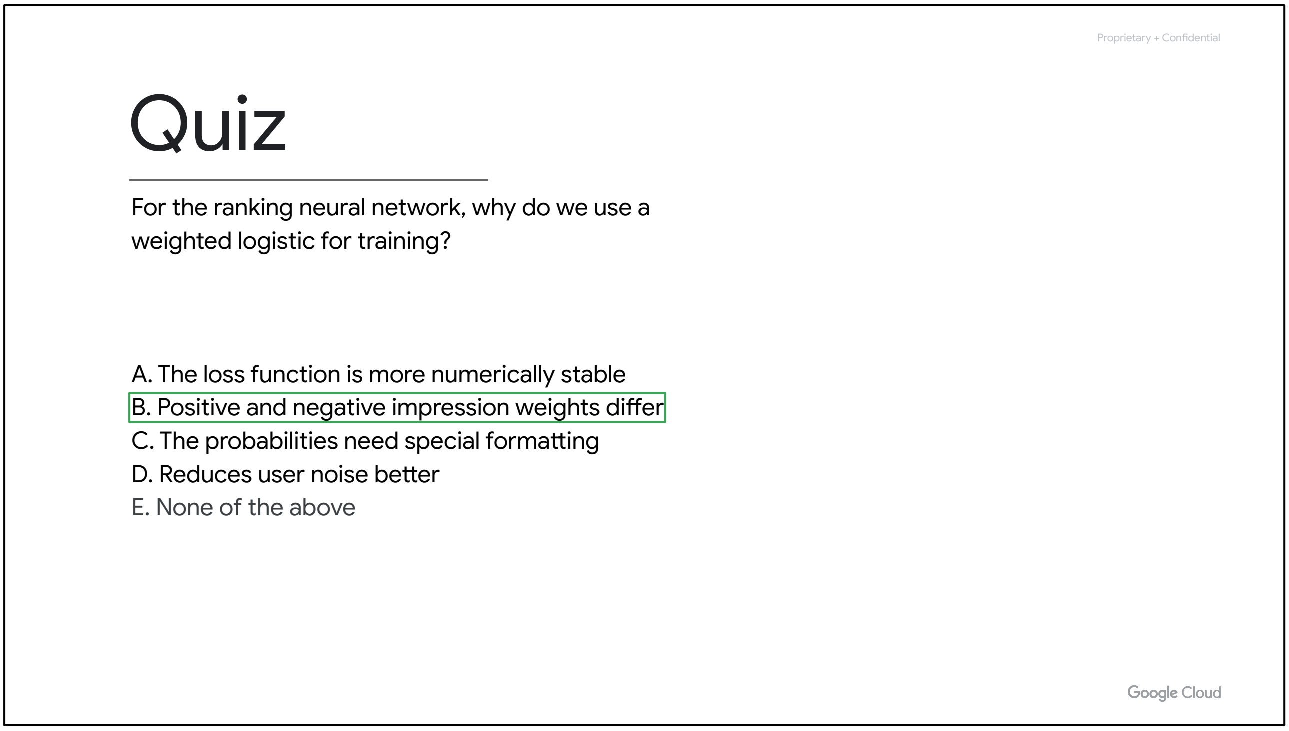


Quiz

For the ranking neural network, why do we use a weighted logistic for training?

A. The loss function is more numerically stable
B. Positive and negative impression weights differ
C. The probabilities need special formatting
D. Reduces user noise better
E. None of the above

Now that we have gone through the second half, the ranking neural network, let's test your knowledge! For the ranking neural network, why do we use a weighted logistic for training? Is it that the loss function is more numerically stable than a normal logistic? Is it that positive and negative impressions need to be weighted differently? Maybe the probabilities need to be formatted in special ways? Maybe it reduces user noise better? Or maybe it's just none of the above?

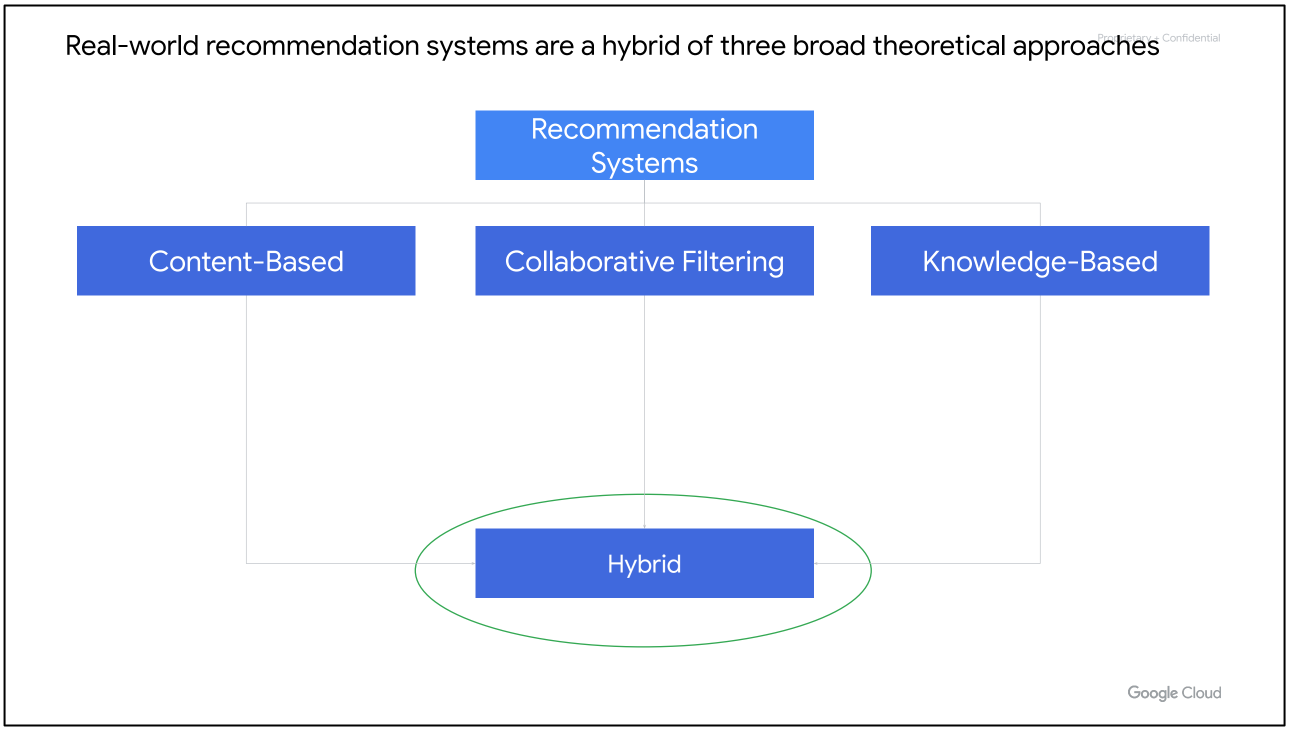


Quiz

For the ranking neural network, why do we use a weighted logistic for training?

A. The loss function is more numerically stable
[✓ B. Positive and negative impression weights differ]
C. The probabilities need special formatting
D. Reduces user noise better
E. None of the above

The correct answer is B! Because the labels are expected watch time, we want to weight videos based on their watch time and also based on their impressions. For positive impressions, we weight the output probabilities by watch time, and for negative impressions, we weight the output probabilities by a unit weight. We don't do this during serving, because our model has already been trained on expected watch time. Furthermore, because these are new videos, the user hasn't watched them, so we don't have the actual watch time to weight the probabilities with.

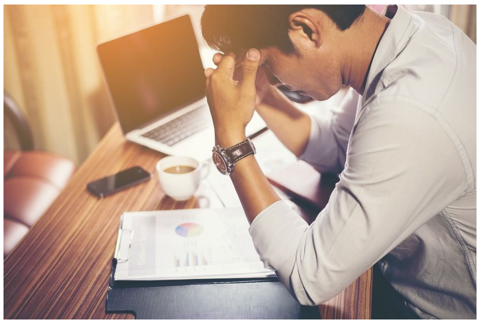


Real-world recommendation systems are a hybrid of three broad theoretical approaches

Using neural networks to combine all of the advantages, and eliminate the disadvantages, of the three broad types of recommendation systems is very powerful. We were able to eliminate, or at least mitigate, the disadvantages of each broad recommendation system approach and use full advantage of their individual strengths. These hybrid models fortunately don't use any fancy new system and just build off of what we already have. However, to get the full power out of them, we should provide as much data as possible from many different sources and connect all of these models together into an ML pipeline.

We put all of this together ourselves, in a hybrid neural network lab.

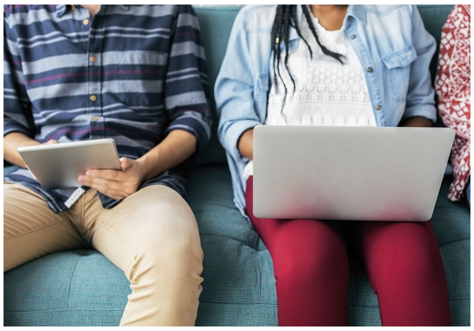
Context components



We learned about how context factors into our experiences with items such as what our mood was at the time, like our stress level.

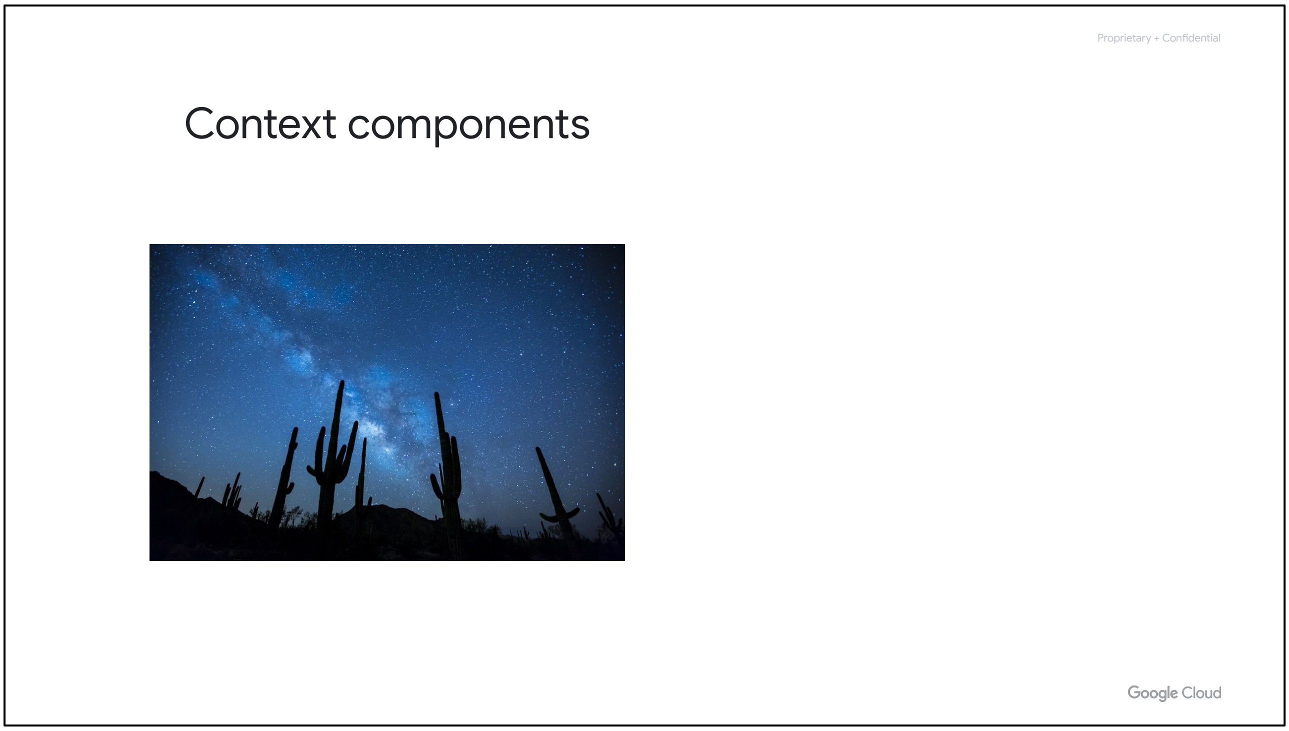
https://pixabay.com/en/work-stressed-accounts-man-working-2005640/ cc0

Context components



Who else we experience things with, like our family.

https://pixabay.com/en/boy-break-browsing-casual-computer-1986107/ cc0



Context components

Where we experience things that matter, from the everyday locations to the truly amazing.

https://pixabay.com/en/milky-way-stars-night-sky-923738/ cc0

Context components

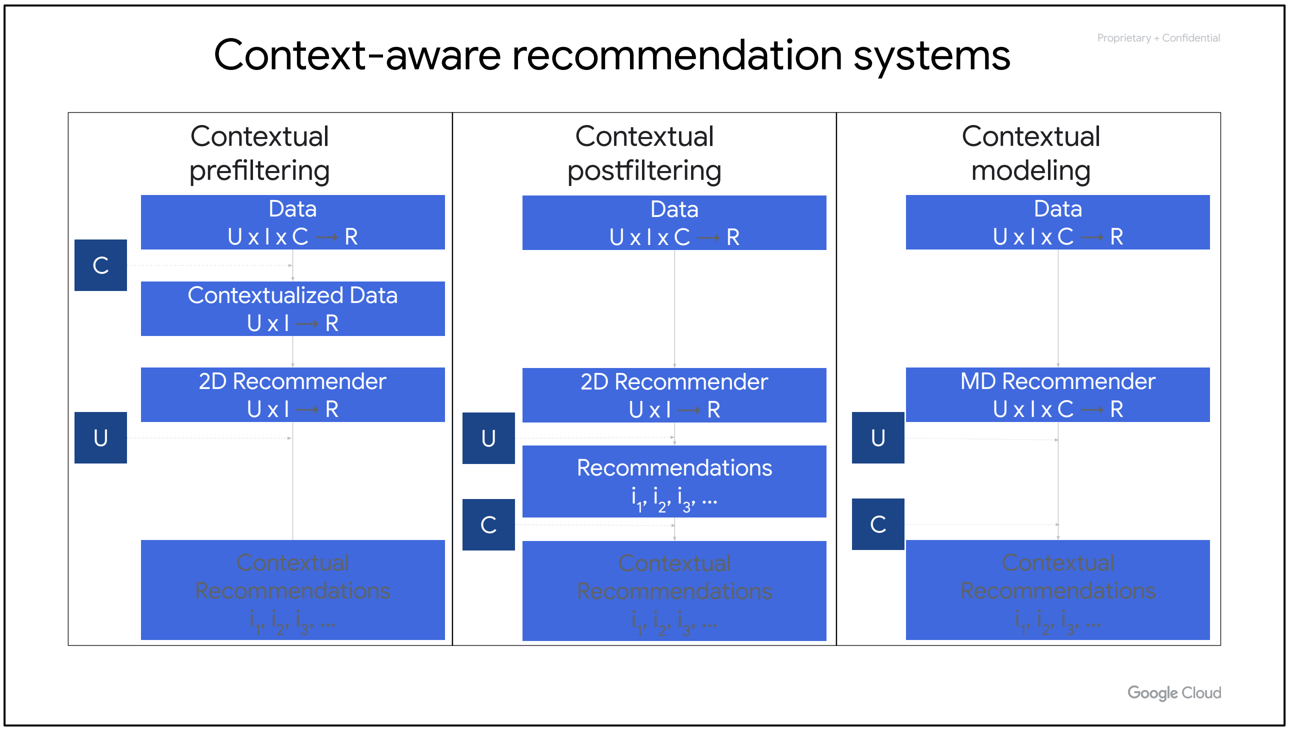
How time plays a factor in our experience, such as the difference in seasons.

https://pixabay.com/en/house-cottage-winter-red-cottage-2151102/ cc0

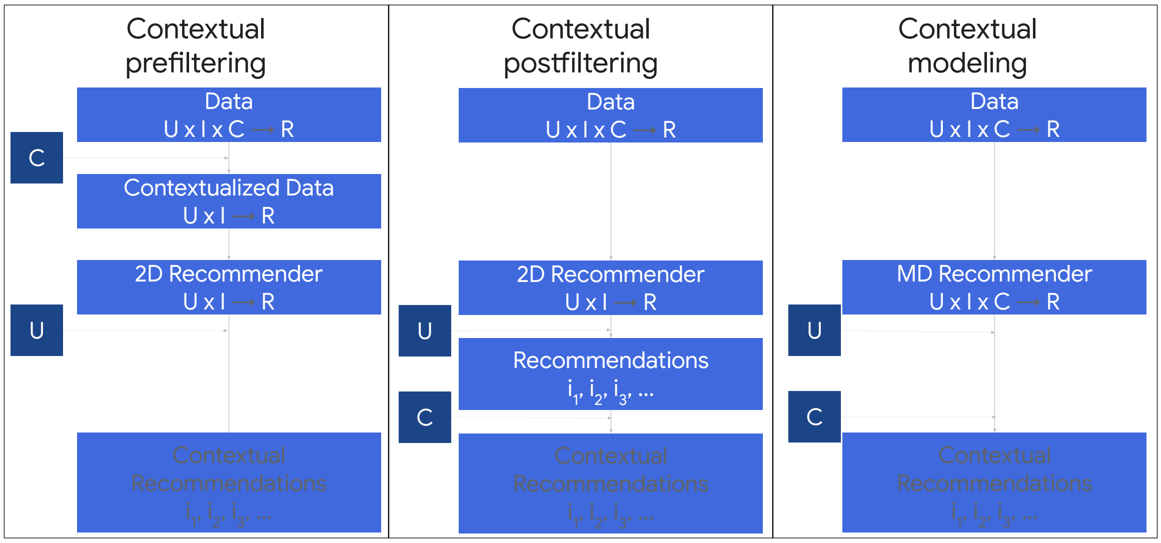
Context components

Or if we are experiencing things during special occasions wrapped up with emotion, excitement, and expectations?

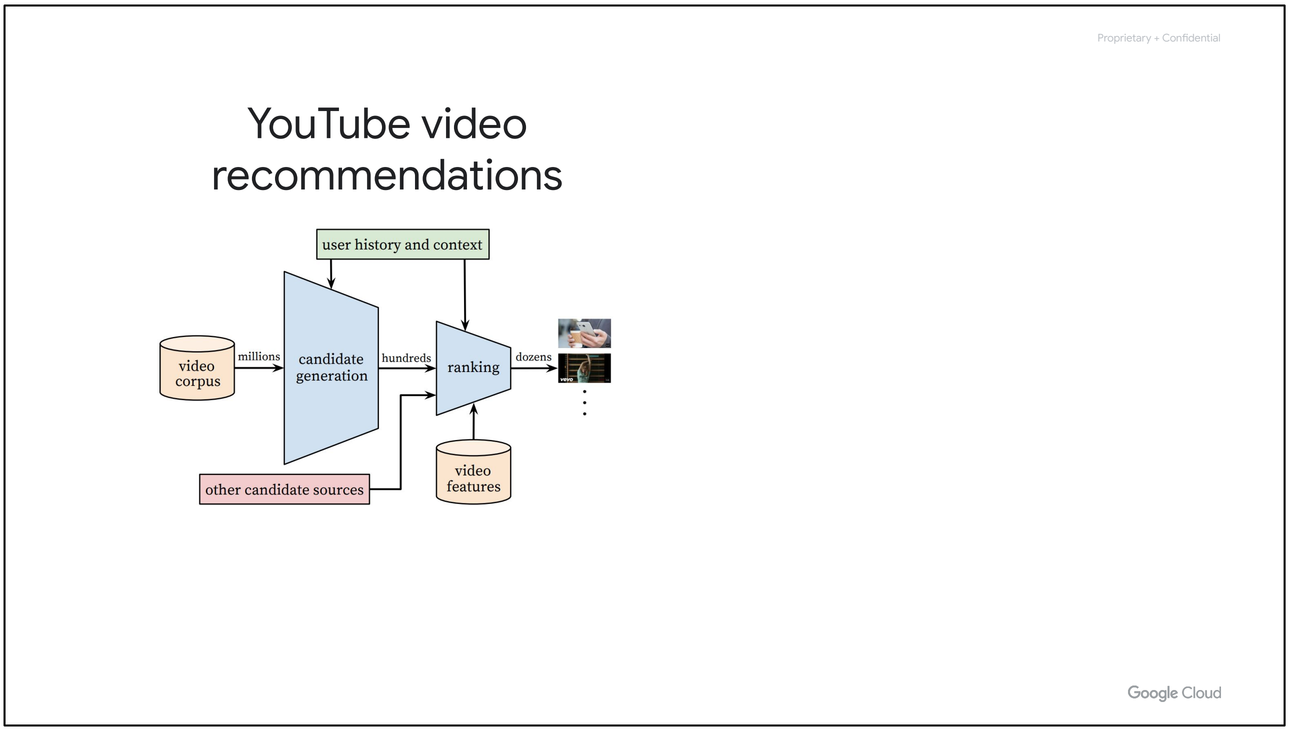
https://pixabay.com/en/restaurant-people-eating-690975/ cc0



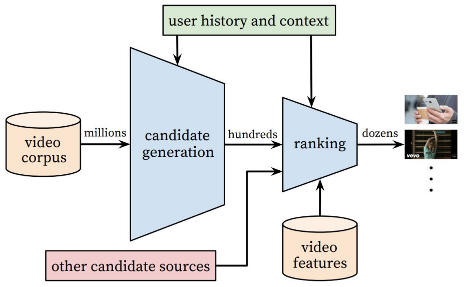
Context-aware recommendation systems



We learned how to take this context and use it with the three main approaches of context-aware recommendation systems: contextual prefiltering, postfiltering, and modeling.



YouTube video recommendations



Lastly, we learned about some of the inner workings of the YouTube video hybrid recommendation system and saw how two different neural networks harnessing enormous amounts of data can create some very powerful recommendations to delight its users.

In the next module, we will see how we can productionize and automate much of the necessary pipeline using the greater Google Cloud Platform ecosystem to make our hybrid neural network recommendation system even better.