



Content-Based Recommendation Systems

Michael Munn

In the last module we mentioned a few common types of recommendation systems. In this module we'll go a bit deeper in exploring how content-based recommendation systems work.

Content-based filtering uses item features to recommend new items that are similar to what the user has liked in the past.

Content-based filtering methods use item features to recommend new items that are similar to what the user has already liked, based on their previous actions or explicit feedback. They don't rely on information about other users or other user-item interactions.

Content-based filtering uses item features to recommend new items that are similar to what the user has liked in the past.



For example, consider the following scenario...

We have a user who has seen and rated a few movies. Some she liked and gave a thumbs up, and some she didn't. We would like to know which movie in our database to recommend next.

Which of the remaining unrated/unseen movies should we recommend to this user?

Quiz

Based on the ratings we currently have, which of the remaining, unrated movies should we recommend to this user?

- a) The Dark Knight Rises
- b) The Incredibles
- c) Bleu



Based on the ratings we currently have, which of the remaining unrated movies should we recommend to this user?

The Dark Knight Rises
The Incredibles
or Bleu

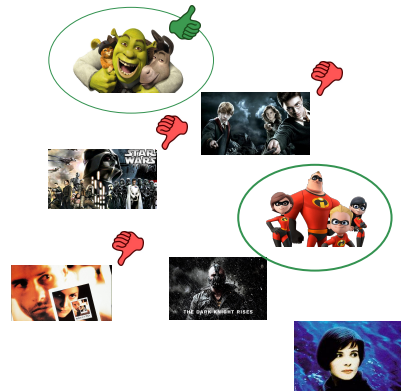
Answer

Based on the ratings we currently have, which of the remaining, unrated movies should we recommend to this user?

a) The Dark Knight Rises

b) The Incredibles

c) Bleu



That's right. We would likely want to recommend The Incredibles.

Given a list of all movies in the corpus, if one movie is similar to another that the user ranked highly, then we'll recommend that one.

We don't have a lot of information to base our recommendation on here. But this user seems to enjoy cartoon movies and not any of the others. Based on the positive rating this user gave to Shrek, and the similarity between Shrek and The Incredibles, perhaps we should recommend The Incredibles to watch next.

Learn how to...

Measure the similarity of elements in an embedding space.

Discuss the mechanics of content-based recommendation systems.

Build a content-based recommendation system.

In this module, we will learn how to...

Learn how to...

Measure the similarity of elements in an embedding space.

Discuss the mechanics of content-based recommendation systems.

Build a content-based recommendation system.

Measure the similarity of pairs of elements in an embedding space.

Learn how to...

Measure the similarity of elements in an embedding space.

Discuss the mechanics of content-based recommendation systems.

Build a content-based recommendation system.

We'll discuss the mechanics of content-based recommendation systems, and really see how they work....

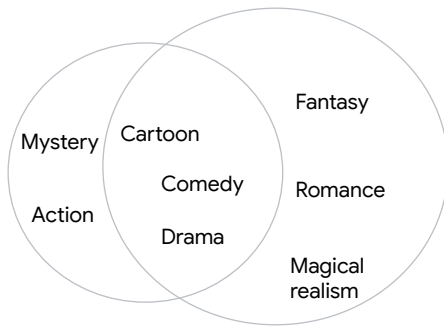
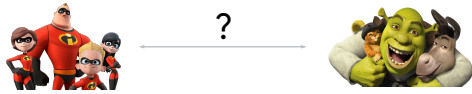
Learn how to...

Measure the similarity of elements in an embedding space.

Discuss the mechanics of content-based recommendation systems.

Build a content-based recommendation system.

...And we'll see how to build our own content-based recommendation systems: one to recommend movies, and one to recommend articles.

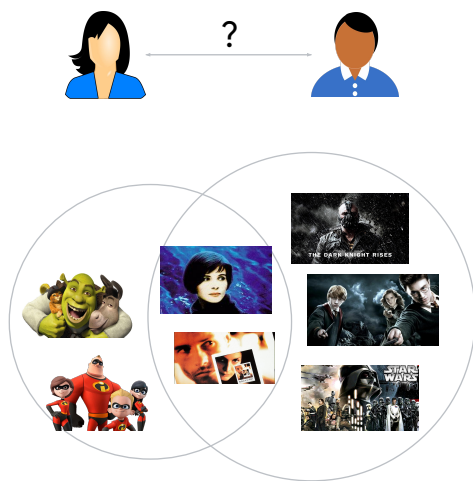


In the quiz we had earlier we had some idea that The Incredibles and Shrek were similar.

But what does it mean for two movies to be similar?

We can try to answer this question in a number of ways.

One way is to consider the different genres or themes a movie has. If there is a lot of overlap in themes between the two movies, then perhaps we can say the movies are similar.

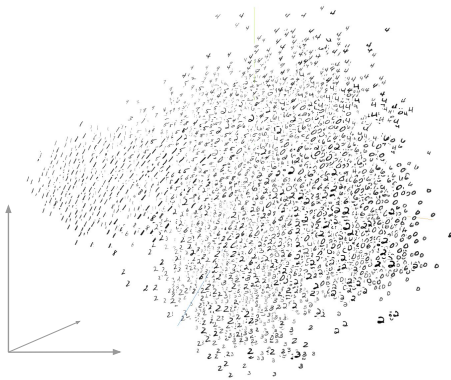


In the same way, we can ask what it means for two **users** to be similar.

We *could* try use the same genre analysis, or another reasonable approach would be to consider the movies they've liked in the past. If two users like a lot of the same movies, we can say that those two users are similar.

To do machine learning, we'll want to compare movies and users, so we need to make this notion of similarity more rigorous. This is often done by thinking of the properties (or features) of items and users in the same embedding space where we can then compare how similar they are.

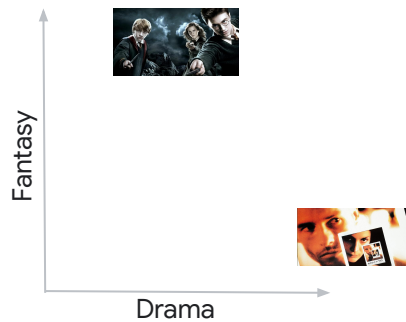
An **embedding** is a map from our collection of items to some finite dimensional vector space.



An *embedding* is a map from our collection of items (or users) to some finite dimensional vector space. It provides a way of giving a finite, vector-valued representation of the items (and users) in our dataset.

Embeddings are commonly used to represent input features in machine learning problems. In fact, we've already seen how embeddings can arise in some of the examples from previous modules. Here is a visualization of the MNIST dataset of handwritten images, embedded into 3-dimensions. If you look closely, this embedding shows how all the 1's lie in the same general area and all the 2's lie together as well, and so on.

(e.g. <http://projector.tensorflow.org> with the MNIST examples. This is an embedding of digits into 3-dimensions)



Let's go back to our movie example and consider a two-dimensional embedding for the elements in our dataset.

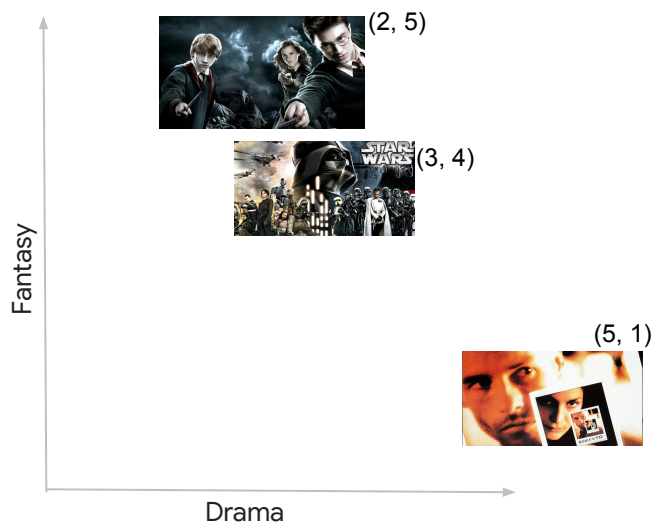
That is, suppose we describe our movies using just two genres (or dimensions). Say, drama and fantasy. We'll put 'drama' on the x-axis and 'fantasy' on the y-axis.

A movie like Memento scores high in the 'drama' genre but has a relatively low value for 'fantasy.'

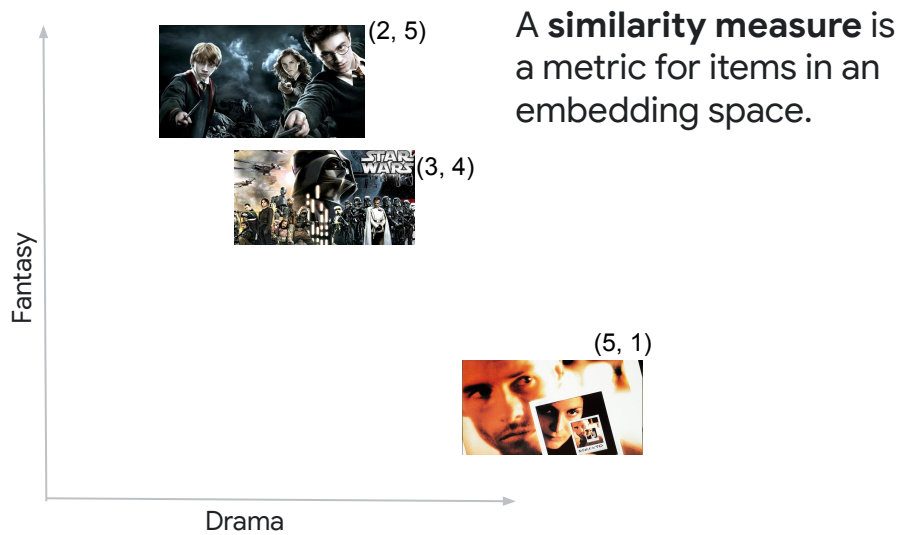
Where as, Harry Potter measures high for "fantasy" but only medium for "drama."



A new movie, like Star Wars, might appear to be closer to Harry Potter in this embedded space.



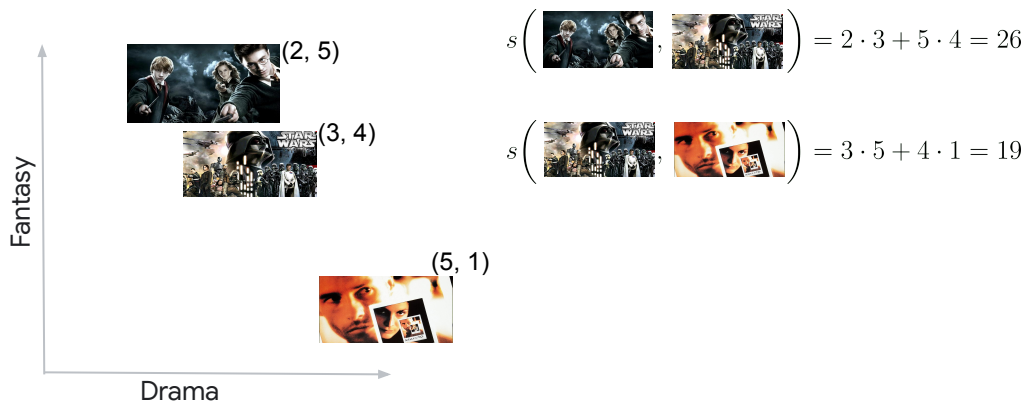
But how close? To make this precise, consider the following embedding values for the movies here.



A **similarity measure** is just a metric that defines exactly how similar, or close, two items are in the embedding space.

One commonly used similarity measure is the *dot product*.

$$\text{dot product: } s(\vec{a}, \vec{b}) = \sum_i a_i b_i$$



To compute the dot product of two vectors, we compute the sum of the product-wise components of each vector.

Because 26 is greater than 19, Star Wars is more similar to Harry Potter than it is to Memento. Which confirms our intuition.

$$s(\text{Harry Potter}, \text{Star Wars}) = 2 \cdot 3 + 5 \cdot 4 = 26$$

$$s(\text{Star Wars}, \text{Memento}) = 3 \cdot 5 + 4 \cdot 1 = 19$$

$$\text{cosine similarity: } s(\vec{a}, \vec{b}) = \frac{\sum_i a_i b_i}{|\vec{a}| |\vec{b}|}$$



The cosine similarity is another popularly used similarity measure.

It is similar to the dot product, but scaled by the norm of the movie feature vectors. Note that the similarity is still greatest between Harry Potter and Star Wars.

$$s(\text{Harry Potter}, \text{Star Wars}) = \frac{(2)(3) + (5)(4)}{\sqrt{29}\sqrt{25}} = 0.97$$

$$s(\text{Star Wars}, \text{Indiana Jones}) = \frac{(3)(5) + (4)(1)}{\sqrt{25}\sqrt{26}} = 0.75$$

Quiz

Compute the cosine similarity between Star Wars and Shrek and between Harry Potter and The Incredibles. Which pair of movies is more similar?



- a) Star Wars and Shrek
- b) Harry Potter and The Incredibles
- c) The pairs are equally similar

Let's have a quick quiz to check our understanding of similarity measures. Compute the cosine similarity between Star Wars and Shrek and between Harry Potter and The Incredibles. Which pair of movies is more similar?

Answer

Compute the cosine similarity between Star Wars and Shrek and between Harry Potter and The Incredibles. Which pair of movies is more similar?

$$s\left(\begin{array}{c} \text{Star Wars} \\ \text{Movie} \end{array}, \begin{array}{c} \text{Shrek} \\ \text{Movie} \end{array}\right) = \frac{(3)(-1) + (4)(5)}{\sqrt{25}\sqrt{26}} = 0.67$$

$$s\left(\begin{array}{c} \text{Harry Potter} \\ \text{Movie} \end{array}, \begin{array}{c} \text{The Incredibles} \\ \text{Movie} \end{array}\right) = \frac{(2)(-3) + (5)(3)}{\sqrt{29}\sqrt{18}} = 0.39$$

a) Star Wars and Shrek

b) Harry Potter and The Incredibles

c) The pairs are equally similar

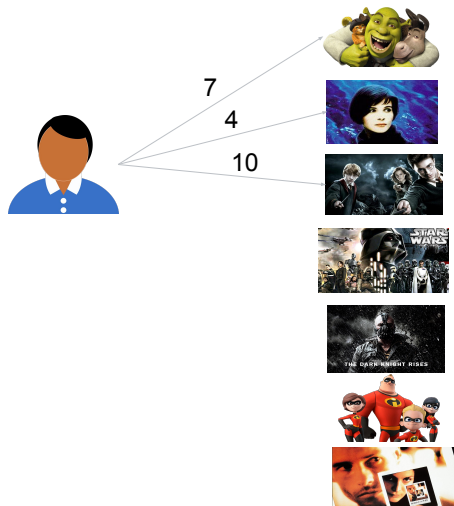
That's right. Star Wars and Shrek are more similar because the cosine similarity measure between them is greater.

Check that these are the values you found when computing the cosine similarity between these pairs of movies.



Let's go a bit deeper and look at a specific example, putting some numbers to the previous ideas to see how exactly we can create a recommendation using content-based filtering.

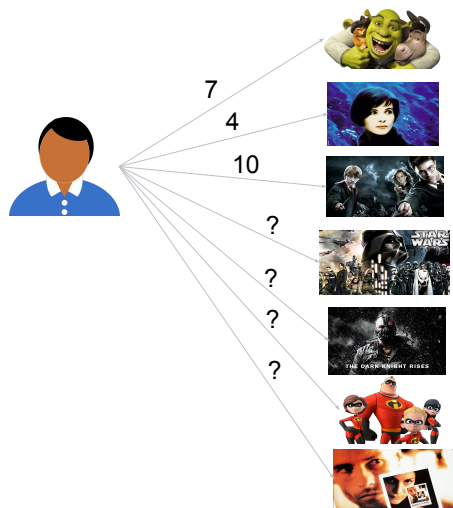
Consider a single user, and suppose we have only 7 movies in our database.



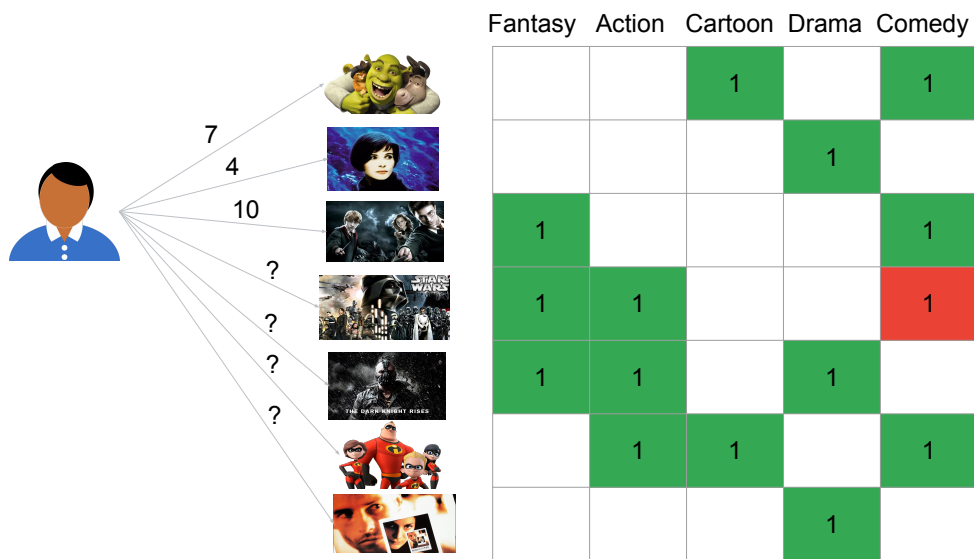
This user has seen and rated 3 of the movies. We'd like to figure out which of the remaining 4 movies to recommend.

We'll assume a rating scale of 1 to 10. One means they didn't like it, and 10 means they loved it.

This user gave Shrek a 7 out of 10, Bleu a 4 out of 10, and Harry Potter a 10 out of 10.



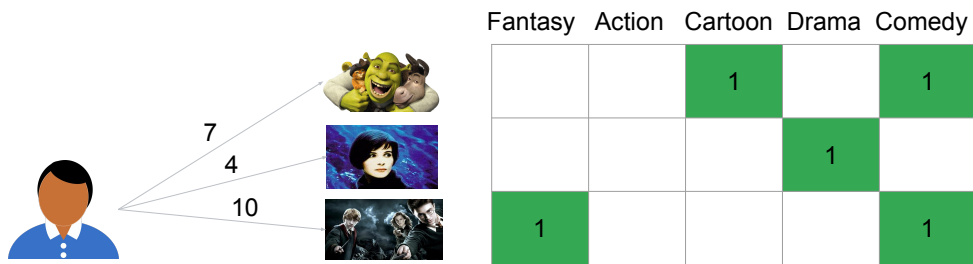
We'd like to use this information to recommend one of the movies the user hasn't seen yet.



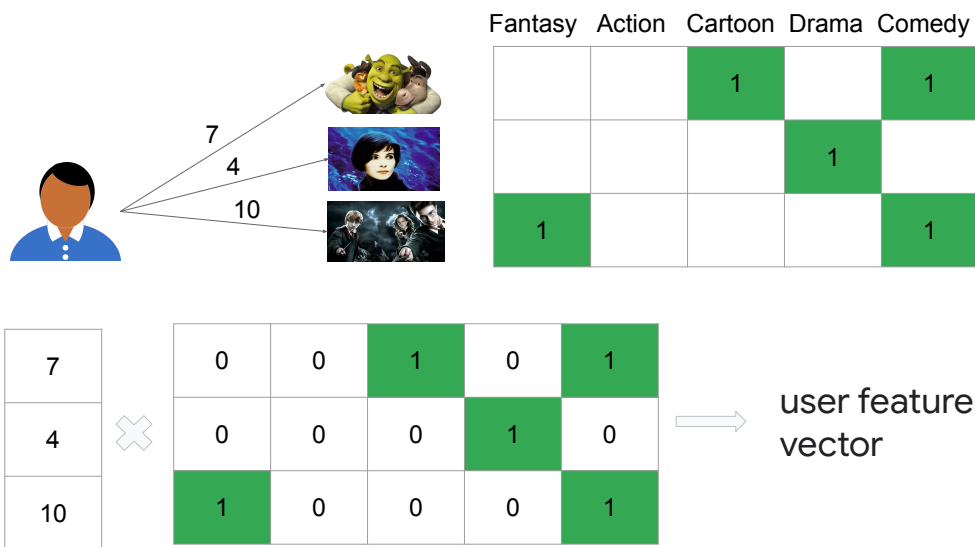
To do this, we represent each of these movies using predetermined features (or genres).

Here we're using the genres Fantasy, Action, Cartoon, Drama, and Comedy. Each movie is k-hot encoded as to whether it has that feature.

Some movies satisfy only one feature; some have more. You can imagine with more granularity of features we'd be able to describe our movies in a more precise way. But for now, we'll use just these five categories.

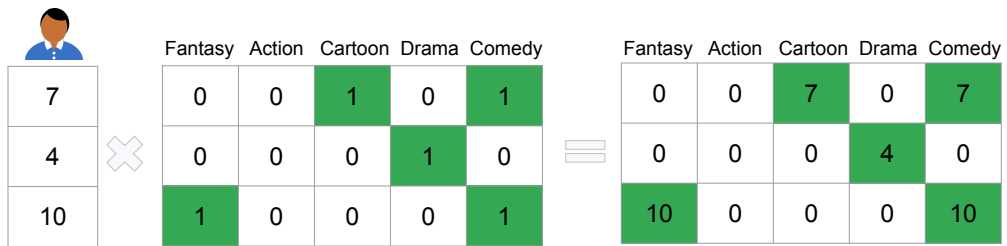


Given their previous movie ratings, we can describe our user in terms of the same features we used to describe our movies. That is, we can place our user in the same 5-dimensional embedded feature space that we are using to represent our movies.



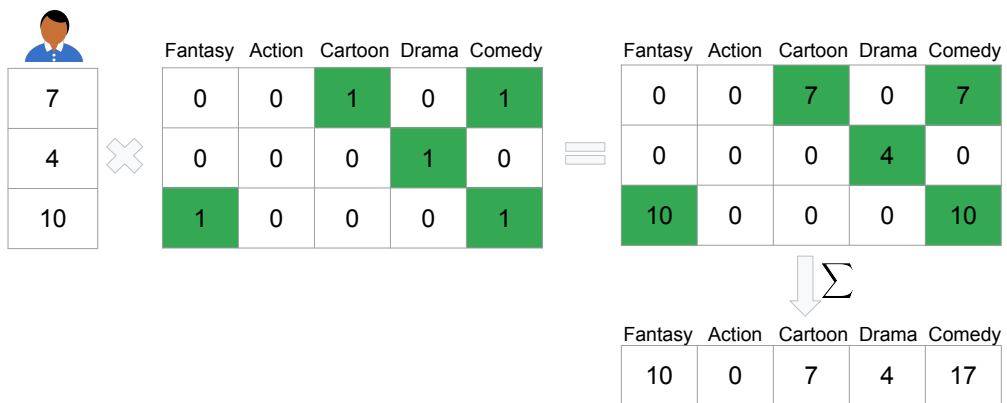
To do this, we first scale each feature by this user's ratings and then normalize the resulting vector. This is called the 'user-feature' vector.

Basically, it gives an idea of where our user sits in our embedding space of features, based on their previous ratings of various movies in our database.



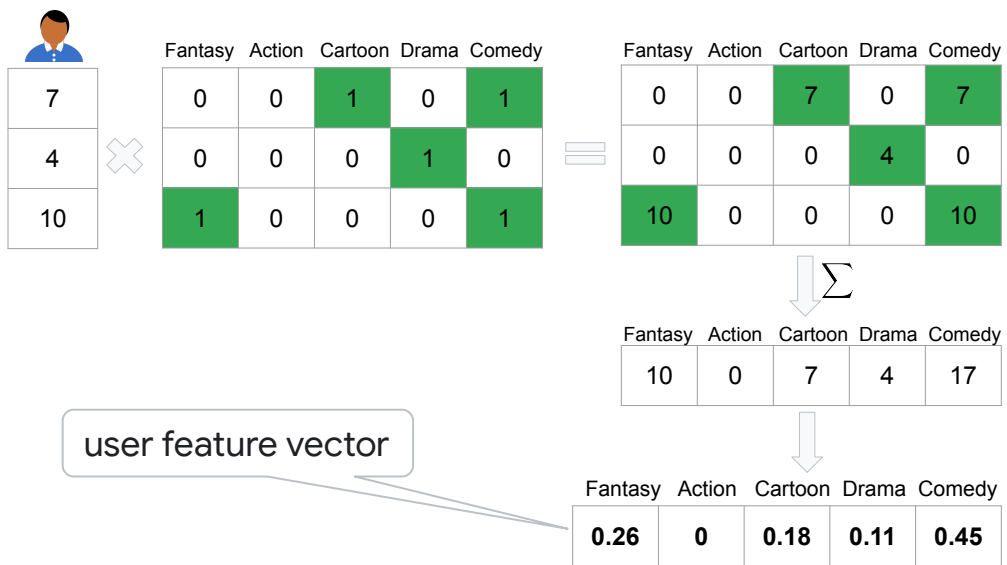
Let's work through that now.

First, multiply the movie feature matrix by the ratings given by that user...

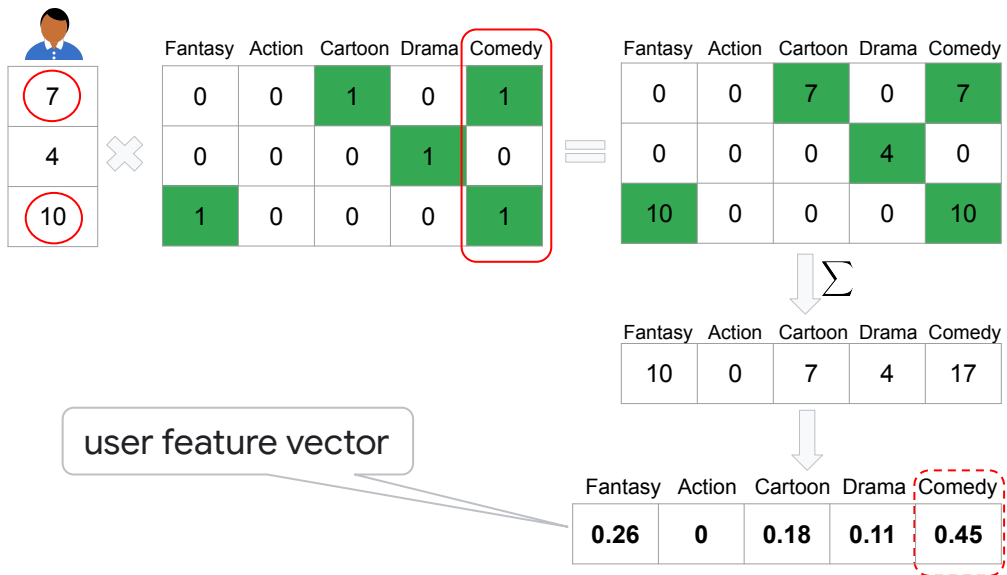


...then aggregate by summing across each feature dimension.

This gives us a 5-dimensional vector in our feature space embedding.

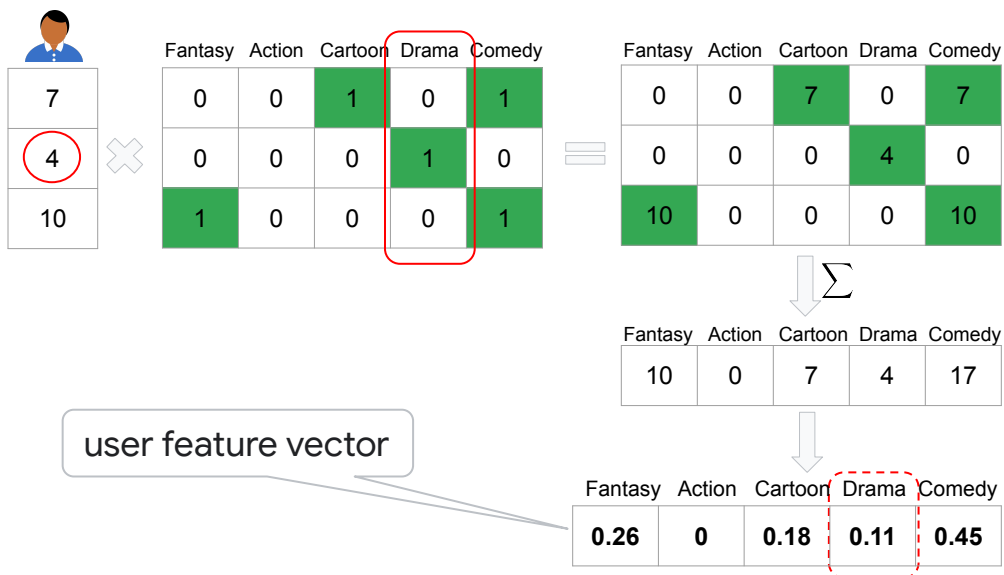


The user feature vector is the normalization of that vector.



We see that for this user, 'Comedy' seems to be a favorite category. It has the largest value. This makes sense looking back at their ratings for the three movies.

The two movies that were classified as "Comedy" have relatively high ratings; 7 out of 10 and 10 out of 10....

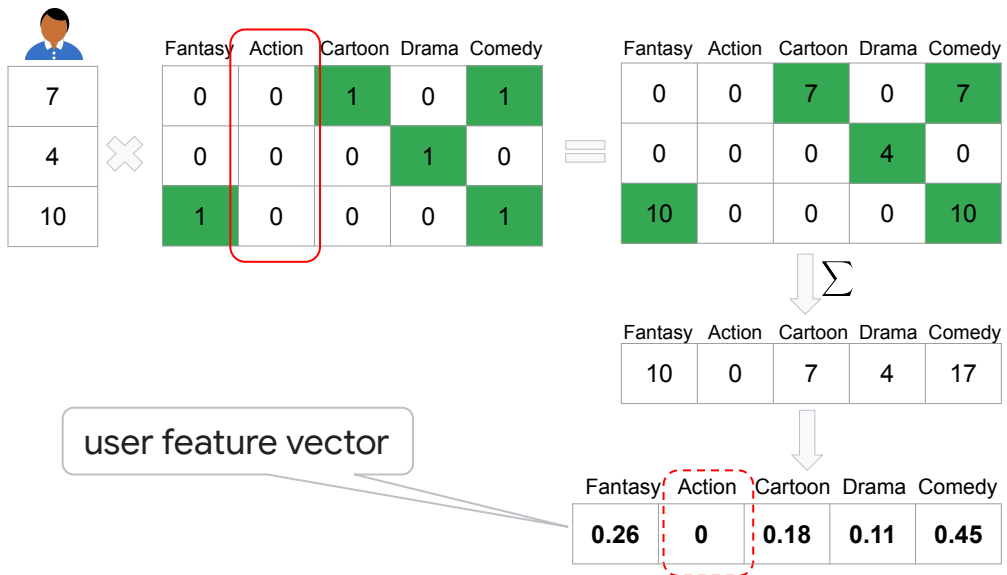


...the "Drama" category appears to be the lowest, which also makes sense looking at the rating of this user. For the one "Drama" movie they have seen, they didn't rate it very highly.

The numeric values of the user feature vector make sense with the intuition we have from the user's ratings and the feature descriptions of the movies.

It is interesting to point out that the 'Action' dimension is zero for this user. Is this because the user doesn't like action at all? Not necessarily.

If you look at their ratings, none of the movies they've previously rated contain the 'Action' feature. Think about how this affects our user-feature vector; we'll come back to this later.



Not necessarily.

If you look at their ratings, none of the movies they've previously rated contain the 'Action' feature. Think about how this affects our user-feature vector; we'll come back to this later.

Quiz

Compute the user feature vector for this user based on the ratings below. Which category has the strongest influence for this user?

- a) Fantasy
- b) Action
- c) Cartoon
- d) Drama
- e) Comedy



8

10

2

6



Fantasy	Action	Cartoon	Drama	Comedy
		1		1
			1	
1				1
1	1			1
1	1		1	
	1	1		1
			1	

Let's look at another user's movie ratings. Compute the user feature vector for this user, based on their ratings for the movies and their respective features.

Which category has the strongest influence for this user?

Answer

Compute the user feature vector for this user based on the ratings below. Which category has the strongest influence for this user?

- a) Fantasy
- b) Action
- c) Cartoon
- d) Drama
- e) Comedy

Fantasy	Action	Cartoon	Drama	Comedy
8	0	0	0	8
10	10	0	10	0
0	2	2	0	2
0	0	0	6	0



18	12	2	16	10
----	----	---	----	----

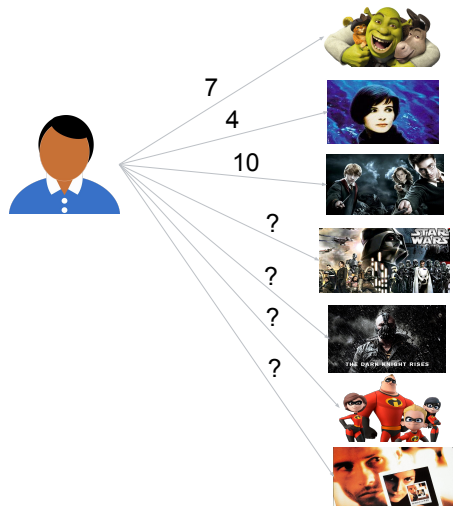


Fantasy	Action	Cartoon	Drama	Comedy
0.31	0.21	0.03	0.28	0.17

For this user, the Fantasy category has greatest value, and thus the strongest influence.

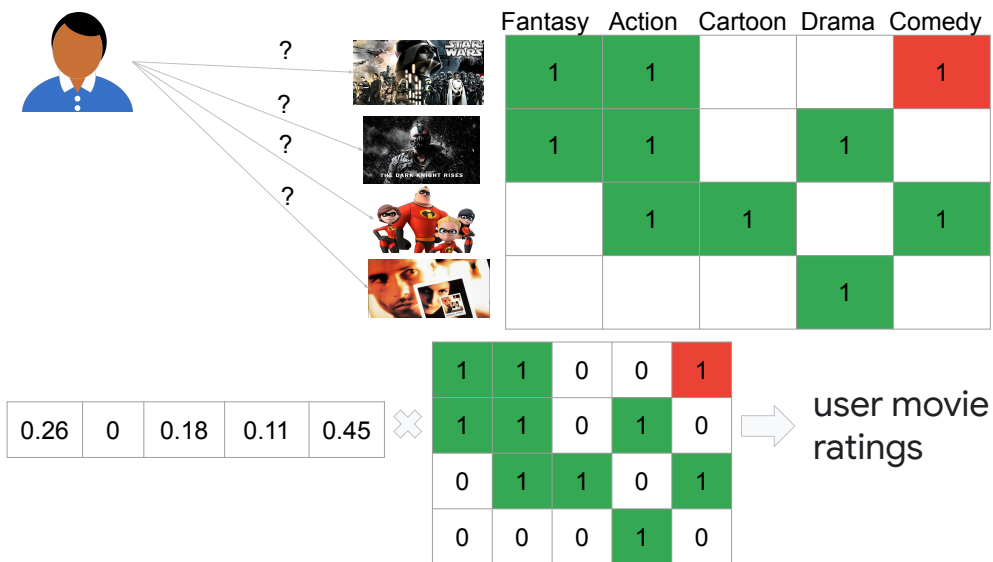
To verify this, first scale the movie-feature matrix by the user's ratings, then sum across the feature columns. The user-feature vector is the normalization of that vector.

In doing this computation, we see that Fantasy has a relative score of 0.31, although Drama is close with 0.28. This means that the Fantasy category has the strongest influence.








Now we can make the best recommendation for our user based on their user-feature vector and the features of the unrated/unseen movies in our database.

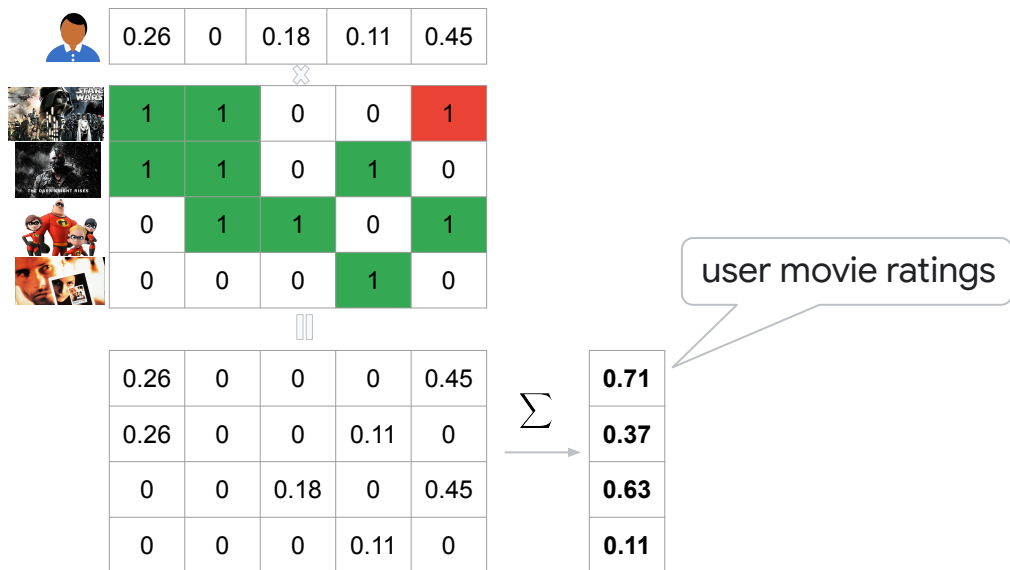
To do this, we'll use a similarity measure like we previously described.



Basically, we'll compute the dot product to measure the similarity between our user and all the remaining unranked movies in our database. The movie with the greatest similarity measure is our top recommendation for our user.






	0.26	0	0.18	0.11	0.45
\otimes					
	1	1	0	0	1
	1	1	0	1	0
	0	1	1	0	1
	0	0	0	1	0
\parallel					
	0.26	0	0	0	0.45
	0.26	0	0	0.11	0
	0	0	0.18	0	0.45
	0	0	0	0.11	0

The dot product is found by taking the component-wise product across each dimension and adding the results.
That is, we multiply the user-feature vector component-wise with the movie-feature vector for each movie...



...then sum row-wise to compute the dot product.

This gives us the dot product similarity between our user and each of the four movies. We'll use these values to make our recommendations

	0.26	0	0.18	0.11	0.45
	1	1	0	0	1
	1	1	0	1	0
	0	1	1	0	1
	0	0	0	1	0

0.26	0	0	0	0.45
0.26	0	0	0.11	0
0	0	0.18	0	0.45
0	0	0	0.11	0

Σ →

0.71
0.37
0.63
0.11

Recommendations



Because Star Wars has the greatest similarity measure, that will be our top recommendation, followed by The Incredibles, and then The Dark Knight, and lastly Memento.

Quiz

In the last quiz, we computed the user-feature vector to be $\langle 0.31, 0.21, 0.03, 0.28, 0.17 \rangle$. Using this, which movie would be our top recommendation for this user?

- a) Shrek
- b) Bleu
- c) Star Wars

In the last quiz, we computed this to be the user-feature vector for that user.

Using this user-feature vector, which movie would be our top recommendation for this user?

Quiz

In the last quiz, we computed the user-feature vector to be $\langle 0.31, 0.21, 0.03, 0.28, 0.17 \rangle$. Using this, which movie would be our top recommendation for this user?

- a) Shrek
- b) Bleu
- c) Star Wars



Fantasy	Action	Cartoon	Drama	Comedy
		1		1
			1	
1	1			1

Remember, these are the movie feature vectors for the three movies.

Answer

In the last quiz, we computed the user-feature vector to be $\langle 0.31, 0.21, 0.03, 0.28, 0.17 \rangle$. Using this, which movie would be our top recommendation for this user?

a) Shrek

b) Bleu





c) Star Wars






Fantasy	Action	Cartoon	Drama	Comedy
		1		1
			1	
1	1			1

The answer here is again Star Wars. Do you know what movie to recommend 2nd?

Answer

	0.31	0.21	0.03	0.28	0.17
	0	0	1	0	1
	0	0	0	1	0
	1	1	0	0	1
	0	0	0.03	0	0.17
	0	0	0	0.28	0
	0.31	0.21	0	0	0.17

 $\xrightarrow{\Sigma}$

	0.20
	0.28
	0.69

To come to that conclusion, we take the dot product of our user-feature vector with each movie. We find that Star Wars has the greatest similarity measure, so it is our top recommendation. Next would be Bleu, and lastly Shrek.

					
	4	6	8		
			10		8
		6			3
	10	9		5	

Ok. So we've seen how content-based filtering can be used to generate movie recommendations for a single user.

We'd like to scale this technique so we can provide recommendations for multiple users at a time.

Here we have a user-movie rating matrix, similar to the user-item interaction matrix we saw in the previous module.

Each row represents a user, and each column represents a movie in our database. The value in row i and column j indicates the rating (from 1 to 10) that user i gave movie j .

Let's walk through the process we previously implemented to see how this would look in theory and how you would implement this in TensorFlow.

					
	4	6	8		
			10		8
		6			3
	10	9		5	

Ok. So we've seen how content-based filtering can be used to generate movie recommendations for a single user.

We'd like to scale this technique so we can provide recommendations for multiple users at a time.






Here we have a user-movie rating matrix, similar to the user-item interaction matrix we saw in the previous module.

Each row represents a user, and each column represents a movie in our database. The value in row i and column j indicates the rating (from 1 to 10) that user i gave movie j .

Let's walk through the process we previously implemented to see how this would look in theory and how you would implement this in TensorFlow.

					
	4	6	8		
			10		8
		6			3
	10	9		5	

user-item rating matrix

	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1			1
	1	1			
			1	1	
	1		1	1	
					1

item-feature matrix

Here is our user-item rating matrix from earlier, next to our item-feature matrix.

Remember, the item-feature matrix gives a k-hot encoding of the features we are using to describe our movies. Each row corresponds to a single movie, and a 1 indicates that the movie fits that genre.

```









# each row represents a user ['Vijay', 'Danielle', 'Ryan', 'Chris']
# each column represents a movie ['Star Wars', 'Dark Knight', 'Shrek',
    'The Incredibles', 'Bleu', 'Harry Potter']
users_movies = tf.constant([[4, 6, 8, 0, 0],
                             [0, 0, 10, 0, 8],
                             [0, 6, 0, 0, 3],
                             [10, 9, 0, 5, 0]])






# the columns represent ['Action', 'Sci-Fi', 'Comedy', 'Cartoon', 'Drama']
movies_feats = tf.constant([[1, 1, 0, 0, 1],
                             [1, 1, 0, 0, 0],
                             [0, 0, 1, 1, 0],
                             [1, 0, 1, 1, 0],
                             [0, 0, 0, 0, 1]])

```

We can initialize these as constants in TensorFlow by creating constant tensor values for our movies and for our movie features.

To do that we'll use `tf.constant` to create two rank-2 tensors with the values for the user-item rating matrix and movie features from before hard-coded.






					
	4	6	8		
			10		8
		6			3
	10	9		5	

	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1			1
	1	1			
			1	1	
	1		1	1	
					1

We want to scale the movie feature matrix by the ratings given by each user.

This will give us a "weighted feature matrix" for each user.

				
	4	6	8	
			10	8
		6		3
	10	9		5






	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1			1
	1	1			
			1	1	
	1		1	1	
					1

4	4	0	0	4
6	6	0	0	0
0	0	8	8	0
0	0	0	0	0
0	0	0	0	0

weighted feature matrix


For the first user, we will get this weighted feature matrix.






					
					
	4	6	8		
			10		8
		6			3
	10	9		5	

	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1			1
	1	1			
			1	1	
	1		1	1	
					1

0	0	0	0	0	4
0	0	0	0	0	0
0	0	10	10	0	0
0	0	0	0	0	0
0	0	0	0	8	0

We repeat this process for the second user...






				
				
	4	6	8	
			10	8
		6		3
	10	9		5

	Action	Sci-Fi	Comedy	Cartoon	Drama
		1	1		1
		1	1		
				1	1
		1		1	
					1

0	0	0	0	0	0
6	6	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	0	0
0	0	0	0	3	3

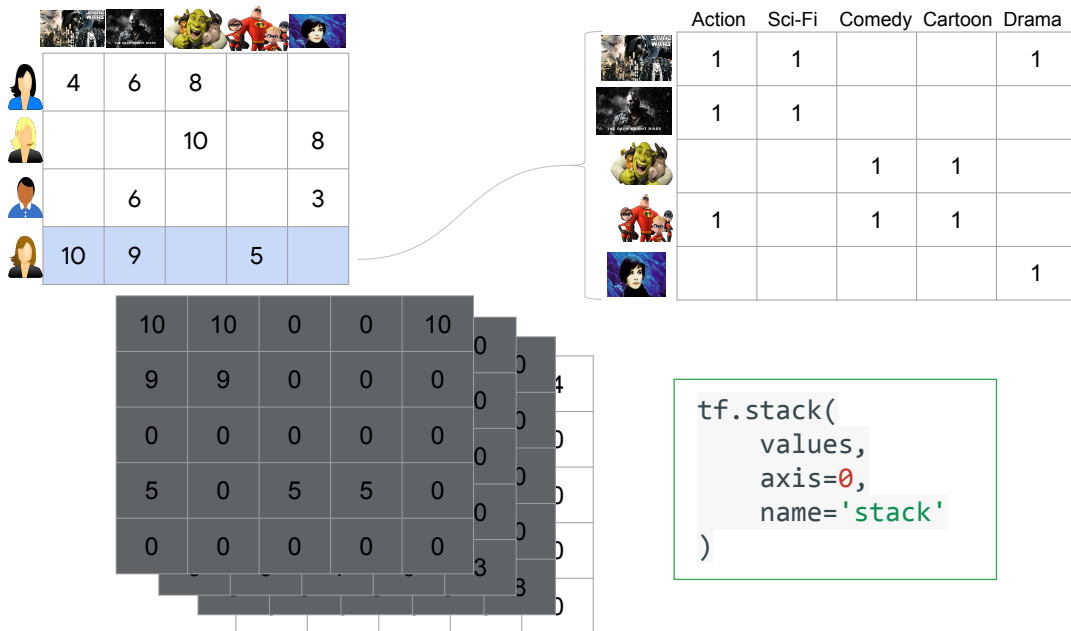
...and the third...

				
				
	4	6	8	
			10	8
		6		3
	10	9		5

	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1			1
	1	1			
			1	1	
	1		1	1	
					1

10	10	0	0	10	0	0	4
9	9	0	0	0	0	0	0
0	0	0	0	0	0	0	0
5	0	5	5	0	0	0	0
0	0	0	0	0	0	0	0
					3	3	0

...and so on.



Once we have this collection of matrices, we can stack them together using `tf.stack` to get a complete weighted user feature tensor.

```
...  
wgted_feature_matrices = [tf.expand_dims(tf.transpose(users_movies)[: ,i],  
                                     axis = 1) *  
                           movies_feats for i in range(num_users)]  
users_movies_feats = tf.stack(wgted_feature_matrices, axis = 0)  
...
```

The previous operations can be done in TensorFlow in the following way:

We first build a list of the weighted feature matrices for each user. Then use `tf.stack` applied to this list, setting the stack axis to be zero.

Quiz

What is the shape of the tensor resulting from stacking together all of the weighted feature matrices?

- a) (# movies, # features)
- b) (# users, # movies, # features)
- c) (# features, #movies, #users)
- d) (# movies, # features, # users)

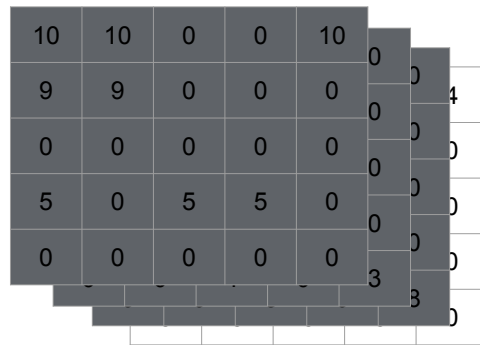
What is the shape of the tensor resulting from stacking together all of the weighted feature matrices?

Answer

What is the shape of the tensor resulting from stacking together all of the weighted feature matrices?

- a) (# movies, # features)
- b) (# users, # movies, # features)
- c) (# features, #movies, #users)
- d) (# movies, # features, # users)

```
tf.stack(wgtd_feature_matrices, axis = 0)
```



The resulting tensor is rank 3, so we can immediately discard option a. Because we stack the weighted feature matrices on axis zero, the shape of this tensor is number of users, by number of movies, by number of features. Or, in this case, 4 by 5 by 5


```
...  
users_movies_feats_sums = tf.reduce_sum(users_movies_feats, axis = 1)  
users_movies_feats_totals = tf.reduce_sum(users_movies_feats_sums, axis = 1)  
  
users_feats = tf.stack([users_movies_feats_sums[i,:]/users_movies_feats_totals[i]  
                        for i in range(num_users)], axis = 0)  
  
...
```

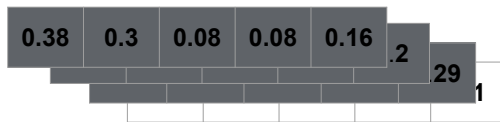
To normalize the users-movies-feats tensor we first sum each column, using `tf.reduce_sum` and setting the axis equal to one. The resulting tensor would then be rank 2 where each row represents the sum of the feature values for each user.





```
...  
users_movies_feats_sums = tf.reduce_sum(users_movies_feats, axis = 1)  
users_movies_feats_totals = tf.reduce_sum(users_movies_feats_sums, axis = 1)  
  
users_feats = tf.stack([users_movies_feats_sums[i,:]/users_movies_feats_totals[i]  
                        for i in range(num_users)], axis = 0)  
  
...
```

Next, we find the feature totals for each user, again using the `tf.reduce_sum` with axis set to one.





The normalization is then just the result of dividing the feature sum by the feature totals for each user. In the end, we stack the resulting tensors together to get the final `users_features` tensor.






user feature tensor




	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16

This results in a user feature tensor, where each row corresponds to a specific user-feature vector.





	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16






	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1


	 0.6				
---	---	---	---	---	---

To find the inferred movie rankings for our users, we compute the dot product between each user-feature vector and each movie-feature vector. In short, we're seeing how similar each user is with respect to each movie, as measured across these 5 feature dimensions.





For example, for our first user, the dot product with Star Wars gives 0.6....






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16




	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1

	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.6	0.5			





...the dot product with The Dark Knight is 0.5.






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16



	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1

					
	0.6	0.5	0.4		





...for Shrek, we get 0.4...






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16







	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1

					
	0.6	0.5	0.4		





...for Shrek, we get 0.4...






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16


	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1

					
	0.6	0.5	0.4	0.65	





...and so on, for the fourth movie....






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16







	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1

					
	0.6	0.5	0.4	0.65	0.1





...and our last movie.






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16

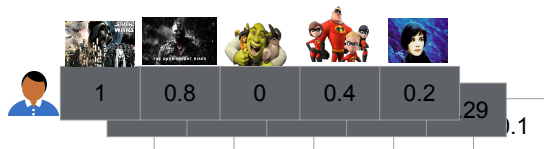
	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1

						0.1
	0.3	0	0.71	0.71	0.29	





We do the same thing for user2...






	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16

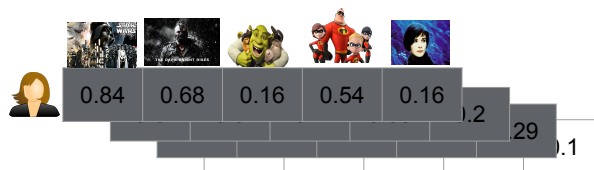
	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1



.....for user3....

	Action	Sci-Fi	Comedy	Cartoon	Drama
	0.25	0.25	0.2	0.2	0.1
	0	0	0.36	0.36	0.29
	0.4	0.4	0	0	0.2
	0.38	0.3	0.08	0.08	0.16

	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1	0	0	1
	1	1	0	0	0
	0	0	1	1	0
	1	0	1	1	0
	0	0	0	0	1



...and finally user4.

```
...  
  
users_ratings = [tf.map_fn(lambda x: tf.tensordot(users_feats[i], x, axes = 1),  
                           tf.cast(movies_feats, tf.float32))  
                  for i in range(num_users)]  
  
all_users_ratings = tf.stack(users_ratings)  
  
...
```

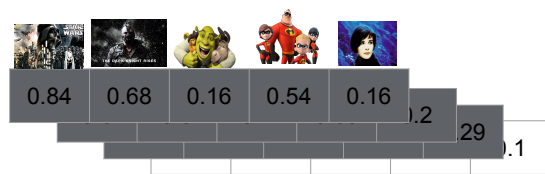
To achieve this in TensorFlow, we can use the map function.

The TensorFlow map function is similar to the map function in numpy. It repeatedly applies some callable function on a sequence of elements from first to last.

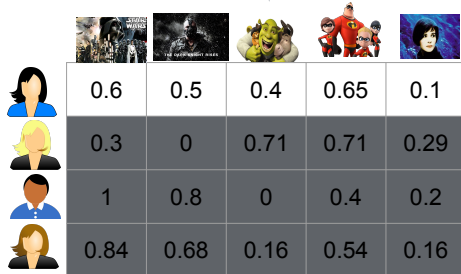
Here we represent that function with lambda; it is simply the dot product of each user-feature vector with each movie-feature vector.

So, the variable `user_ratings` holds a list of the resulting movie rankings for each user and each movie.

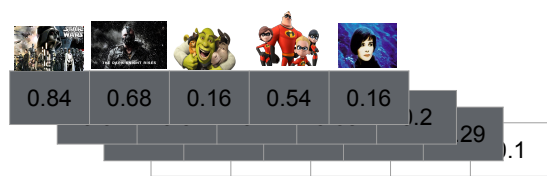
We stack them together to get a tensor of all the users and their movie rankings.



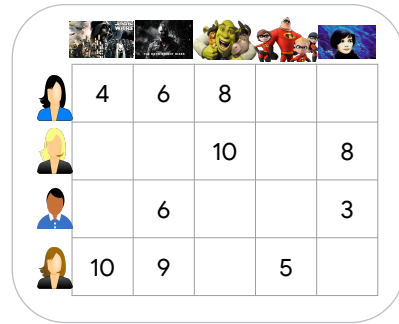
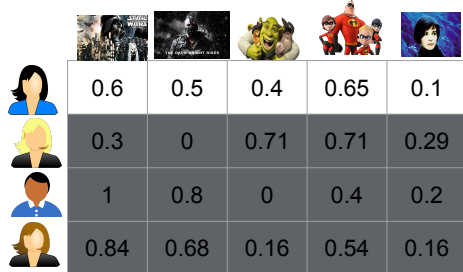
`tf.stack(...)`



Once we have the user-movie ranking matrix we can compare it with the original user-movie matrix to see which movies to recommend to which user.



`tf.stack(...)`



Because our users have already seen and rated some of our movies, we want to mask the ranking for previously rated movies and focus only on unrated or unseen movies for each user.

Quiz

Which TensorFlow operation could we use to mask the previously rated movies in our user-movie ranking matrix, so we only focus on previously unrated movies when providing recommendations?

- a) `tf.mask`
- b) `tf.strided_slice`
- c) `tf.tile`
- d) `tf.sparse_slice`
- e) `tf.where`

Which TensorFlow operation could we use to mask the previously rated movies in our user-movie ranking matrix, so we only focus on previously unrated movies when providing recommendations?

Look through the TensorFlow documentation to see which operation you could use [here](#).

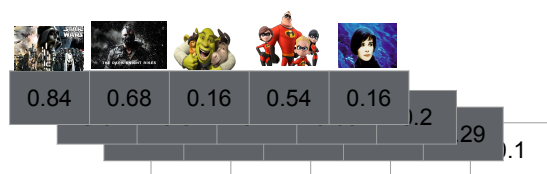
Answer

Which TensorFlow operation could we use to mask the previously rated movies in our user-movie ranking matrix, so we only focus on previously unrated movies when providing recommendations?

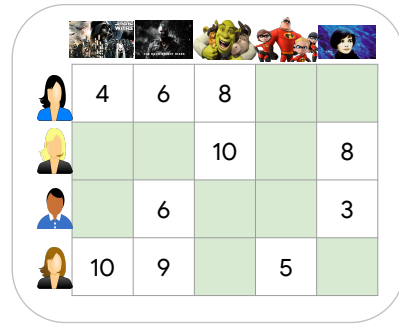
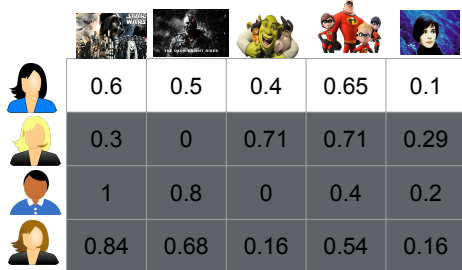
- a) `tf.mask`
- b) `tf.strided_slice`
- c) `tf.tile`
- d) `tf.sparse_slice`
- e) `tf.where`

That's right, `tf.where` operates like numpy's *where* operation. It returns elements based on the value of a condition.

What condition would we use to mask out movies that do not have a rating? Think about it. We'll see the answer in the next slide.



`tf.stack(...)`

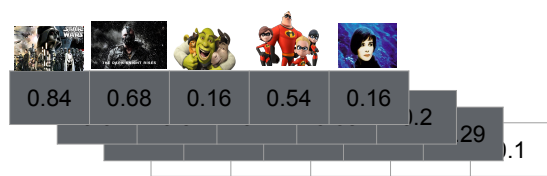


```
tf.where(
    condition,
    x=None,
    y=None,
    name=None
)
```

As we saw in the quiz, we can accomplish this with `tf.where`.

Here the condition variable is a boolean, and `tf.where` will return either the tensor `x` or `y`, depending on the value of the condition.

By setting our condition to be where the user-item interaction matrix does not have values, we can return only those rankings for previously unrated movies.












`tf.stack(...)`

	-	-	-	0.65	0.1
	0.3	0	-	0.71	-
	1	-	0	0.4	-
	-	-	0.16	-	0.16

	4	6	8		
			10		8
		6			3
	10	9		5	

```
tf.where(
    condition,
    x=None,
    y=None,
    name=None
)
```










This results in this user-ranking matrix.

					
				0.65	0.1
	0.3	0		0.71	
	1		0	0.4	
			0.16		0.16

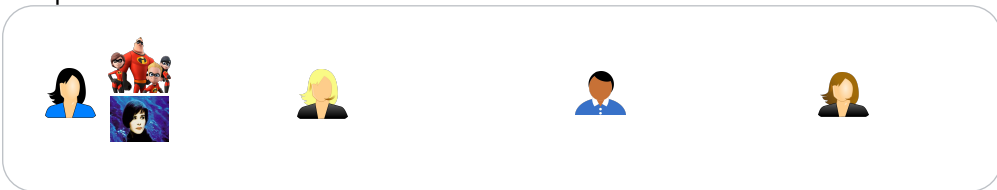
Top Recommendations












Finally, we can use the similarity rankings we've computed here to suggest new movies for each user.

					
				0.65	0.1
	0.3	0		0.71	
	1		0	0.4	
			0.16		0.16

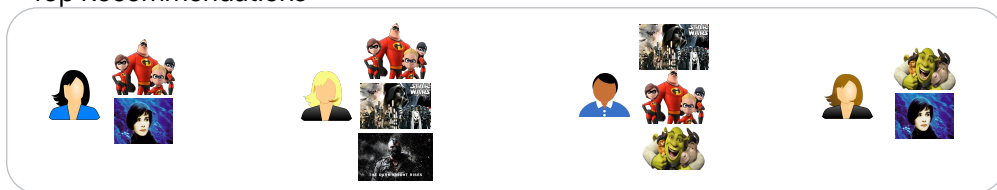
Top Recommendations












For example, for user1, The Incredibles has a higher similarity score than Bleu, so our new movie recommendation list for user1 looks like this.

					
				0.65	0.1
	0.3	0		0.71	
	1		0	0.4	
			0.16		0.16

Top Recommendations












We can also do the same thing for all the other users.






					
				0.65	0.1
	0.3	0		0.71	
	1		0	0.4	
			0.16		0.16

Before we move on, let's take a closer look at our second user.

She has a rating of zero for The Dark Knight. Why is that?

					
				0.65	0.1
	0.3	0		0.71	
	1		0	0.4	
			0.16		0.16

					
	4	6	8		
			10		8
		6			3
	10	9		5	

	Action	Sci-Fi	Comedy	Cartoon	Drama
	1	1			1
	1	1			
			1	1	
	1		1	1	
					1

If we look at the original user-movie rating matrix and compare with the movie feature matrix, we can see why.

Because this user has not rated anything containing 'Action' or 'Sci-Fi,' and the features of the The Dark Knight are solely 'Action' and 'Sci-Fi,' our recommender system will infer a rating of zero for that movie.

This highlights one of the drawbacks of content-based recommender systems: it can be difficult to expand the interests of a user.

If the user hasn't rated movies within one of our predefined features, our recommender won't suggest new movies in that genre.

In this sense, content-based recommenders aren't good at expanding the interests of users to new domains.

Lab

Create a content-based
recommendation system in
TensorFlow

Now that we have learned the basic implementation of content-based filtering, let's put our knowledge into action. This lab demonstrates how to build a content-based recommender using Tensorflow. We'll put code to the multiple user method we just discussed, using most of the code we just saw. You'll see how this can all be done using only low-level Tensorflow operations.

$$f\left(\begin{matrix} \text{user} \\ \text{features}, \end{matrix} \begin{matrix} \text{movie} \\ \text{features} \end{matrix}\right) \overset{?}{\rightarrow} \text{star rating}$$

$$f\left(\begin{matrix} \text{user} \\ \text{features}, \end{matrix} \begin{matrix} \text{movie} \\ \text{features} \end{matrix}\right) \overset{?}{\rightarrow} \text{movie id}$$

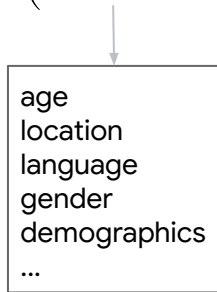
In addition to the tensor or matrix operation approach we saw previously, we can also develop a content-based approach using tools from our supervised machine learning toolbox.

That is, given a user's features and a movie's features, we can try to model the star rating that user might give the movie.

Or, from a classification perspective, perhaps we will try to determine which movie in our database the user will want to watch next.

$$f\left(\overset{\text{user}}{\text{features}}, \overset{\text{movie}}{\text{features}}\right) \overset{?}{\rightarrow} \text{star rating}$$

$$f\left(\overset{\text{user}}{\text{features}}, \overset{\text{movie}}{\text{features}}\right) \overset{?}{\rightarrow} \text{movie id}$$

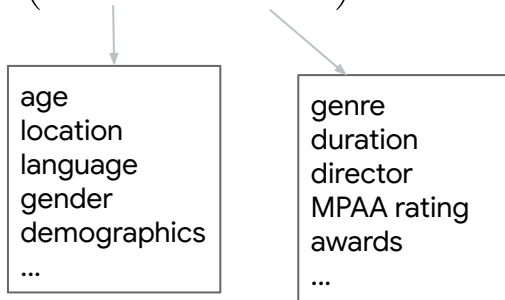


In either case, we would probably want to use features of our user and features of our items (in this case, movies) to make that prediction.

For the user features, we could perhaps consider things like the user's age, location, native language, gender, or other demographics that we think might be relevant.

$$f\left(\overset{\text{user}}{\text{features}}, \overset{\text{movie}}{\text{features}}\right) \overset{?}{\rightarrow} \text{star rating}$$

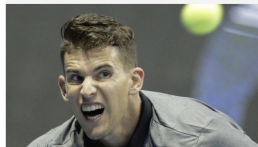
$$f\left(\overset{\text{user}}{\text{features}}, \overset{\text{movie}}{\text{features}}\right) \overset{?}{\rightarrow} \text{movie id}$$



For the movie features, it might be helpful to consider the movie's genre and duration, or the director, its rating with the Motion Picture Association, or whether it's won any awards.

kurier.at

Politik ▾ Regional ▾ Wirtschaft ▾ Sport ▾ Stars Kultur Lifestyle ▾ Video ▾ MEHR ▾



Die Kanzlerin und ihre verunsicherte Partei



Polnische Regierung ruft Auslandspolen zum Denunzieren auf



Bis zu 10.000 Geschädigte durch Bitcoin-Firma



Am 24. Mai enden 24 Jahre Häupl



Olympia-Telegramm: Tag 6 im Rückblick

Let's see how this might work with an example.

We'll use data collected from Kurier.at, a large news provider in Austria.

Here is a snapshot of their website. The center is the front page, and the related items recommendations are on the right.

We'll build a content-based filtering model to recommend articles to readers.

kurier.at

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

The data for this lab is available on BigQuery as part of the cloud-training-demos dataset.

After some SQL queries, we can clean up the initial table and create the following dataset that we will use for training our recommendation model.
Each row of this dataset corresponds to a single visitor interaction with the website.

kurier.at

features

visitor_id	content_id	category		title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis		574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz		574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz		574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer		574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer		574	299814775

The input features of our recommender model pertain to the features of the current article being read.

For example, we will use the current article `content_id` (an ID that corresponds to a single article) and the category, title, author and newness of the current article.

The newness of the current article is captured by the '`months_since_epoch`' column.

The `visitor_id` is actually a `browser_id`, so a different device means a different ID, so we won't use it for our model.

kurier.at

label

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

The goal of our model is to predict the next article read by a visitor. This is represented by the `next_content_id` column.

kurier.at

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

```
content_id_column = tf.feature_column.categorical_column_with_hash_bucket(  
    key="content_id",  
    hash_bucket_size= len(content_ids_list))  
embedded_content_column = tf.feature_column.embedding_column(  
    categorical_column=content_id_column,  
    dimension=10)
```

Let's see how we can incorporate each of these features in TensorFlow.

The content_id is prescribed by Kurier.at and corresponds to the ID of the current article being read. We will make this a categorical column and use hash buckets, because the number of content_ids will likely change over time.

We'll then use an embedding column to embed this categorical feature before passing it off to the model.

kurier.at

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

```
category_column_categorical =  
    tf.feature_column.categorical_column_with_vocabulary_list(  
        key="category",  
        vocabulary_list=categories_list,  
        num_oov_buckets=1)  
category_column = tf.feature_column.indicator_column(category_column_categorical)
```

Similarly for the category column. This is also prescribed by Kurier, though here we have a vocabulary list so, naturally, we use the feature column for categorical features with a vocabulary list.

kurier.at

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

```
embedded_title_column = hub.text_embedding_column(  
    key="title",  
    module_spec="https://tfhub.dev/google/nnlm-de-dim50/1",  
    trainable=False)
```

For the title, we will use a TensorFlow Hub to embed the current article title. In specifying the `module_spec`, we can prescribe which hub module to use; this will add the corresponding module to the TensorFlow graph we are building for our model.

kurier.at

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

```
author_column = tf.feature_column.categorical_column_with_hash_bucket(  
    key="author",  
    hash_bucket_size=len(authors_list) + 1)  
embedded_author_column = tf.feature_column.embedding_column(  
    categorical_column=author_column,  
    dimension=3)
```

Because we have a finite number of authors when training the model, we'll treat the author just as we did the content_id, using categorical columns with hash buckets.

kurier.at

visitor_id	content_id	category	title	author	months_since_epoch	next_content_id
1000196974485173657	299925700	Lifestyle	Nach Tod von Vater: Tochter bekommt jedes Jahr...	Marlene Patsalidis	574	299972194
1000196974485173657	299972194	News	LIVE: Spielstand bei Sturm - Admira	Mathias Kainz	574	299816215
1000196974485173657	299972194	News	Dominante Grazer nehmen Admira auseinander	Mathias Kainz	574	299410466
1007505561418545529	299407839	Stars & Kultur	Trump: 165 Millionen Dollar für 5 Tage Urlaub	Elisabeth Spitzer	574	299816215
1017855659516706306	298846345	Stars & Kultur	Meghan Markle: Lottogewinn veränderte ihr Leben	Elisabeth Spitzer	574	299814775

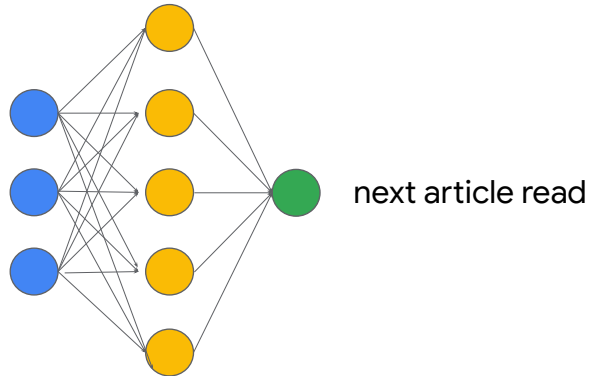
```
months_since_epoch_column = tf.feature_column.numeric_column(  
    key="months_since_epoch")  
months_since_epoch_bucketized = tf.feature_column.bucketized_column(  
    source_column = months_since_epoch_column,  
    boundaries = months_since_epoch_boundaries)
```

The `months_since_epoch` column represents how new the current article is. We'll use this feature as bucketized values.

Of course now that we have these basic feature columns in place, we can add features using crossed columns. For example, perhaps it makes sense to cross the `months_since_epoch` column with the categorical column, because some categories are more or less relevant depending on how new they are.

kurier.at

current article id
current article title
current article author
current article category
current article age
.....



```
net = tf.feature_column.input_layer(features, params['feature_columns'])
for units in params['hidden_units']:
    net = tf.layers.dense(net, units=units, activation=tf.nn.relu)
# Compute logits (1 per class).
logits = tf.layers.dense(net, params['n_classes'], activation=None)
```

The final step is to build our model. We will start with just a single dense layer, but feel free to experiment and try more interesting architectures.

Given the features we have constructed in the previous slides, in code it would look like this.

Lab

Create a content-based recommendation system using a neural network

Michael Munn

Now it's time for you to give it a try. In this lab, we'll build a content-based recommender using a neural network to recommend the next article to read for visitors of the Kurier website.

First, we'll see how to use BigQuery to collect information from the Kurier website Google Analytics data and build our training and test sets. Then, you'll use this data to construct the feature columns that we'll feed into our neural network, including crossed columns combining some of our features. We'll also create additional metrics to assess the performance of our model. And pass this metric to the Tensorboard so we can monitor our performance.

Pros:

Doesn't need information about other users
Can recommend niche items

Let's wrap up by discussing the pros and cons of content-based filters.

Content-based filters are good because they don't need information about other users. Remember, they make predictions based only on that user's interactions with items in the past. This makes them easier to scale.

Another advantage is that they can recommend niche items that other users might not be interested in. Because they rely only on the preferences of that user and items, they do a good job finding lesser known items that the user is likely to enjoy.

Pros:

Doesn't need information about other users
Can recommend niche items

Cons:

Requires domain knowledge to hand-engineer features
Difficult to expand interests of user

They do have their cons, however.

For example, because of the mechanics of building a content-based filter, it is necessary to construct the features you plan to use for the embedding space. It can become difficult to find the most suitable features for more complicated problems.

Also, as we saw in the earlier example, with content-based filters it is difficult to expand the interests of the user outside of what they already know. In this sense, these recommendation systems become dependent on the user's interests.

Quiz

Which of the following are true of content-based recommendation systems? (choose all that apply)

- a) They rely on all user-item interactions.
- b) They use the item features to recommend items for a user, based on items that user has liked in the past.
- c) They are the best filtering method for providing recommendations.
- d) They require hand-engineered features.
- e) They don't do a good job expanding the interests of the user.

Let's end this section with a short quiz.

Which of these statements are true of content-based recommendation systems?
There could be more than one right answer.

Answer

Which of the following are true of content-based recommendation systems?

- a) They rely on all user-item interactions.
- b) They use the item features to recommend items for a user, based on items that user has liked in the past.
- c) They are the best filtering method for providing recommendations.
- d) They require hand-engineered features.
- e) They don't do a good job expanding the interests of the user.

That's right. b) d) and e) are correct.

Content-based filters do not rely on all user-item interactions. They only rely on the item features of items previously liked or rated by a single user. And although they do a good job, it's probably not fair to say they are the best filtering method out there.

They do, however, require hand-engineered features. This is a bit of a drawback actually and we'll see in the next module on collaborative filtering how this can be overcome.

It is also true that content-based filters do not do a good job of expanding the interests of the user. In fact, we saw in an example that if a user hasn't interacted with items containing certain features, a content-based recommender will not be able to start recommending items with solely that feature. So, they don't do a good job expanding the interests of users.

Summary

Mechanics of content-based
recommendation system

How to build your own content-based
recommendation system

In this module, we've done a deep dive into content-based filtering. These are recommendation systems that use item features to recommend new items based on a user's previous actions or explicit feedback. They don't rely on information about other users and their user-item interactions.

We've seen the details of the mechanics of these recommendation filters work, and we've seen them in action through some explicit examples. We've also completed a couple of labs where you developed your own content-based recommendation system.

In the next module you'll learn about another very popular type of recommendation technique: collaborative filtering. This method address some of the drawbacks we saw arise with content-based filters, so they are a very useful tool to know.