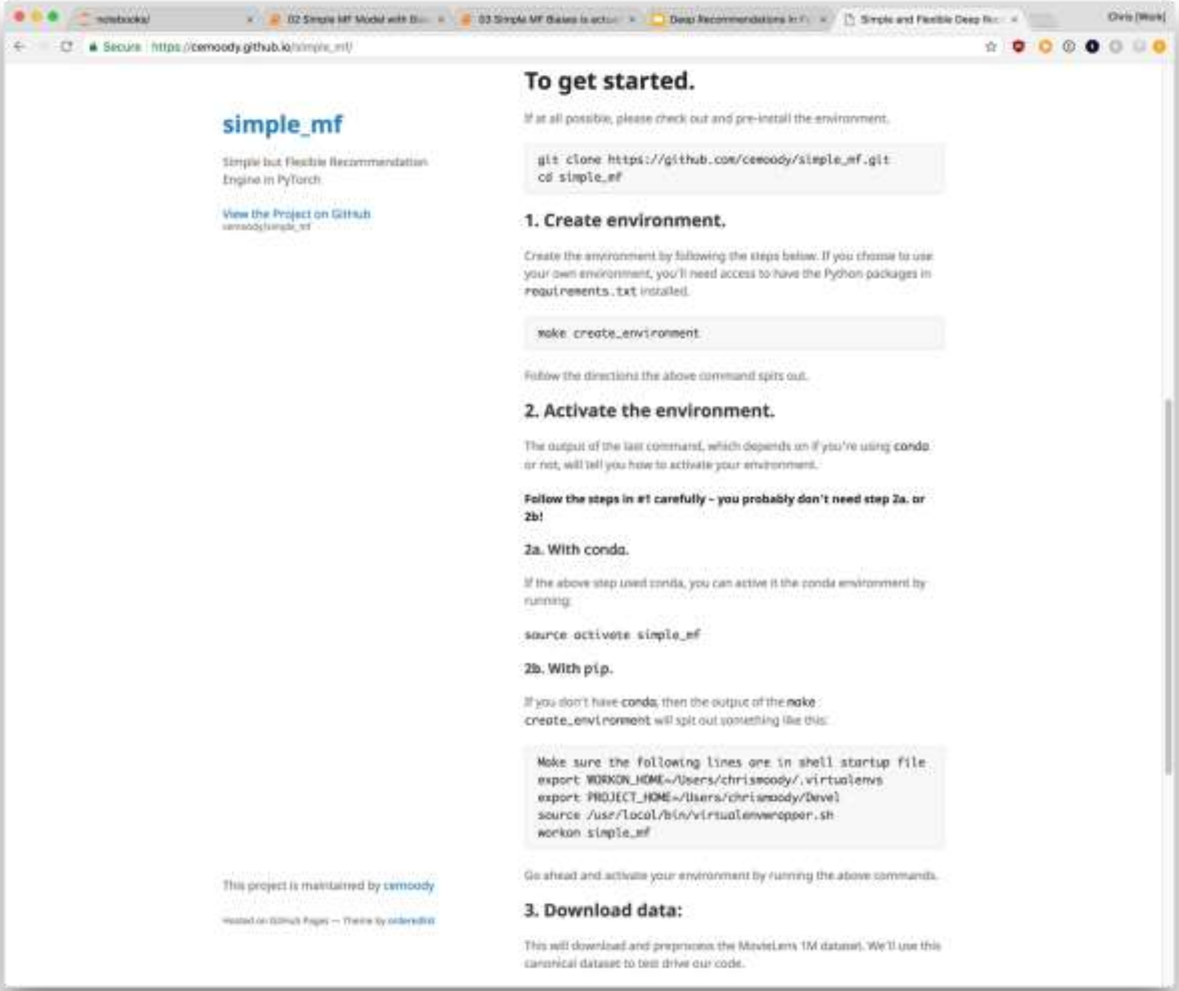
Deep Recommendations in PyTorch

Setup

Follow along with instructions here:

cemoody.github.io/simple_mf



About

@chrisemoody

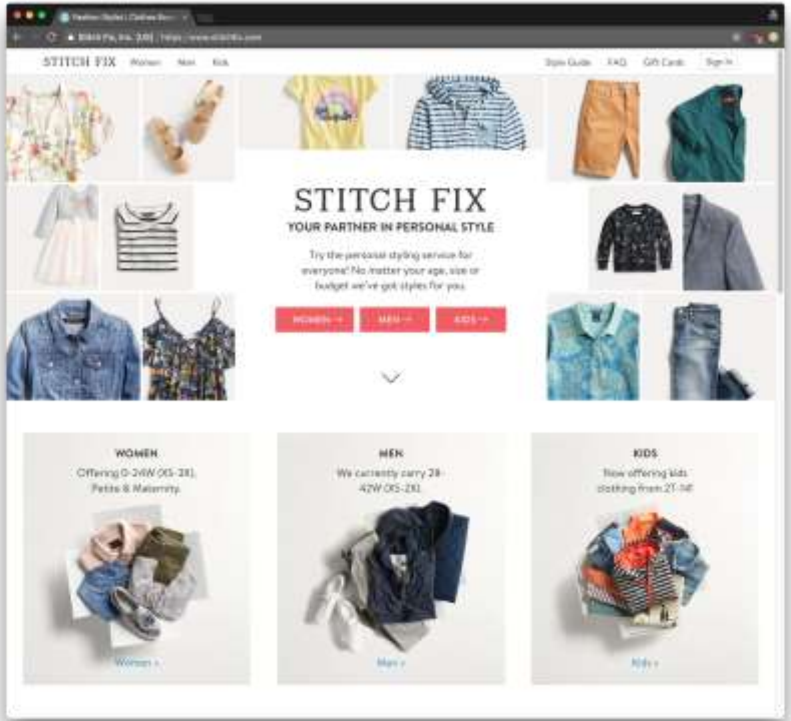
Caltech Physics

PhD. in Astrophysics + Supercomputing

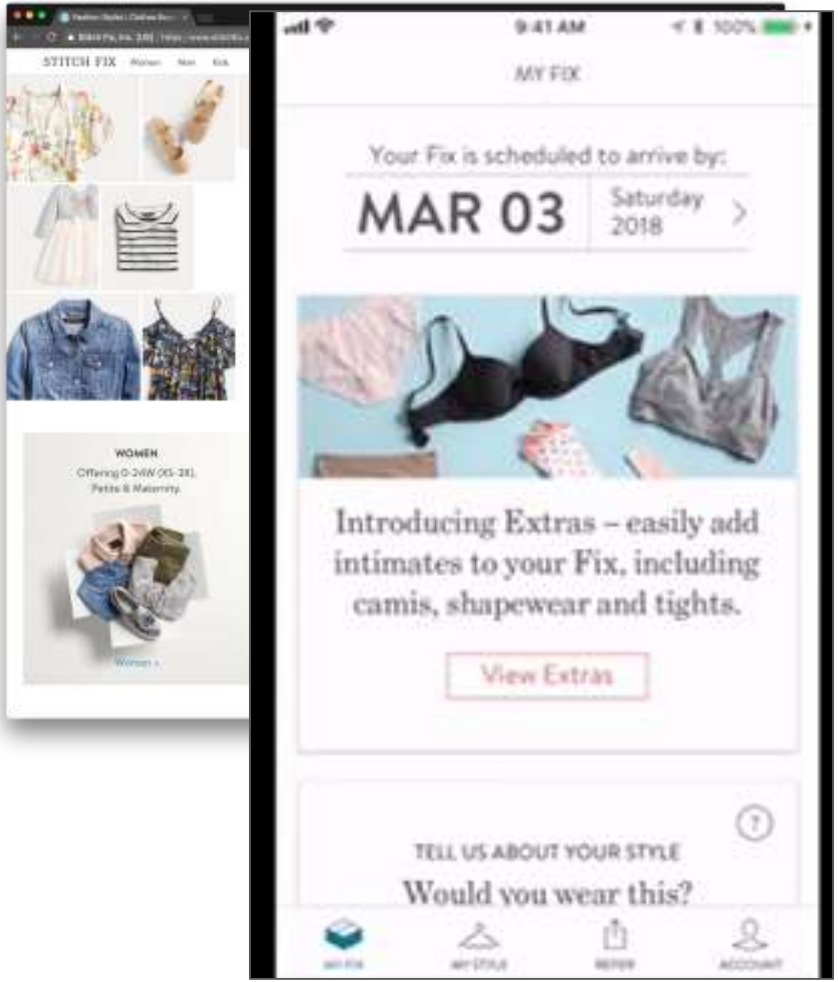
Scikit-Learn t-SNE Contributor

Stitch Fix AI Team

Stitch Fix

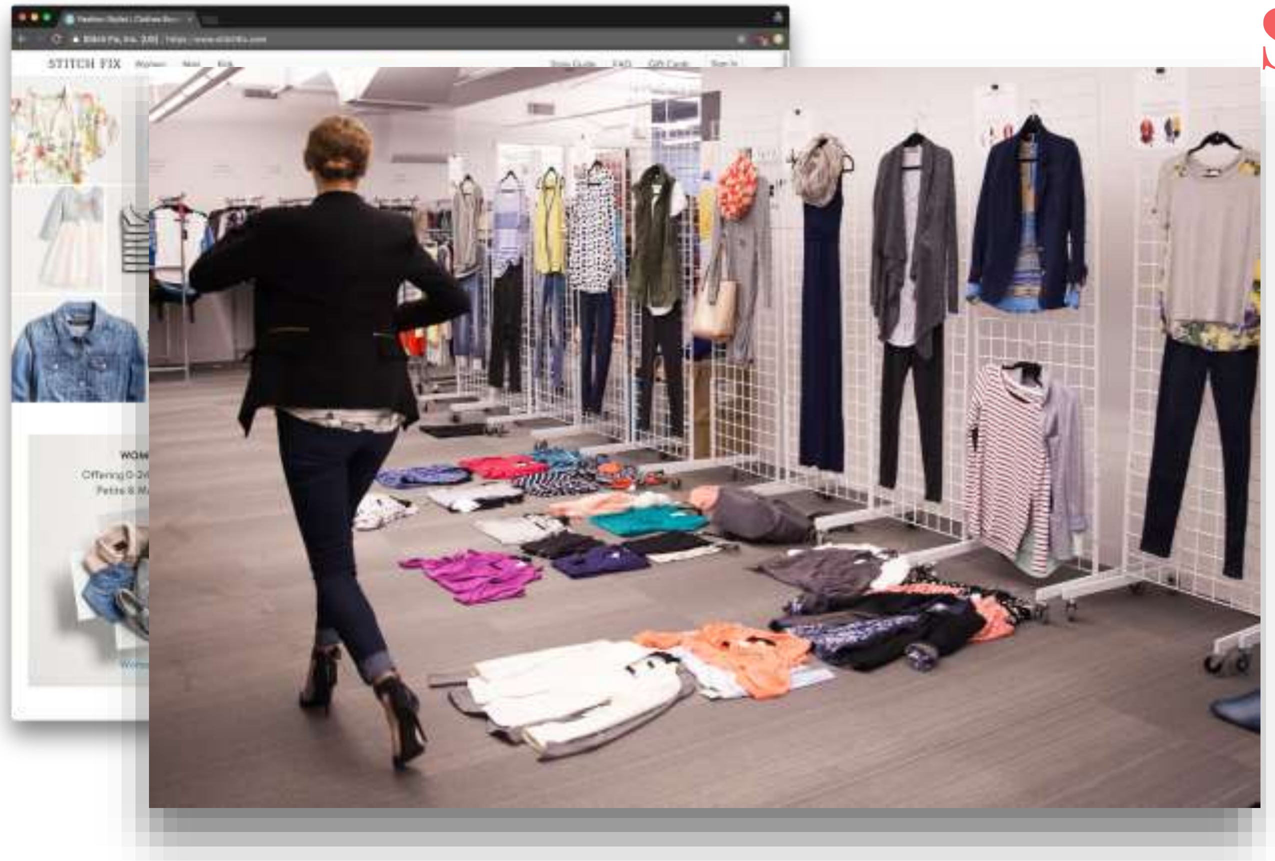


Stitch Fix

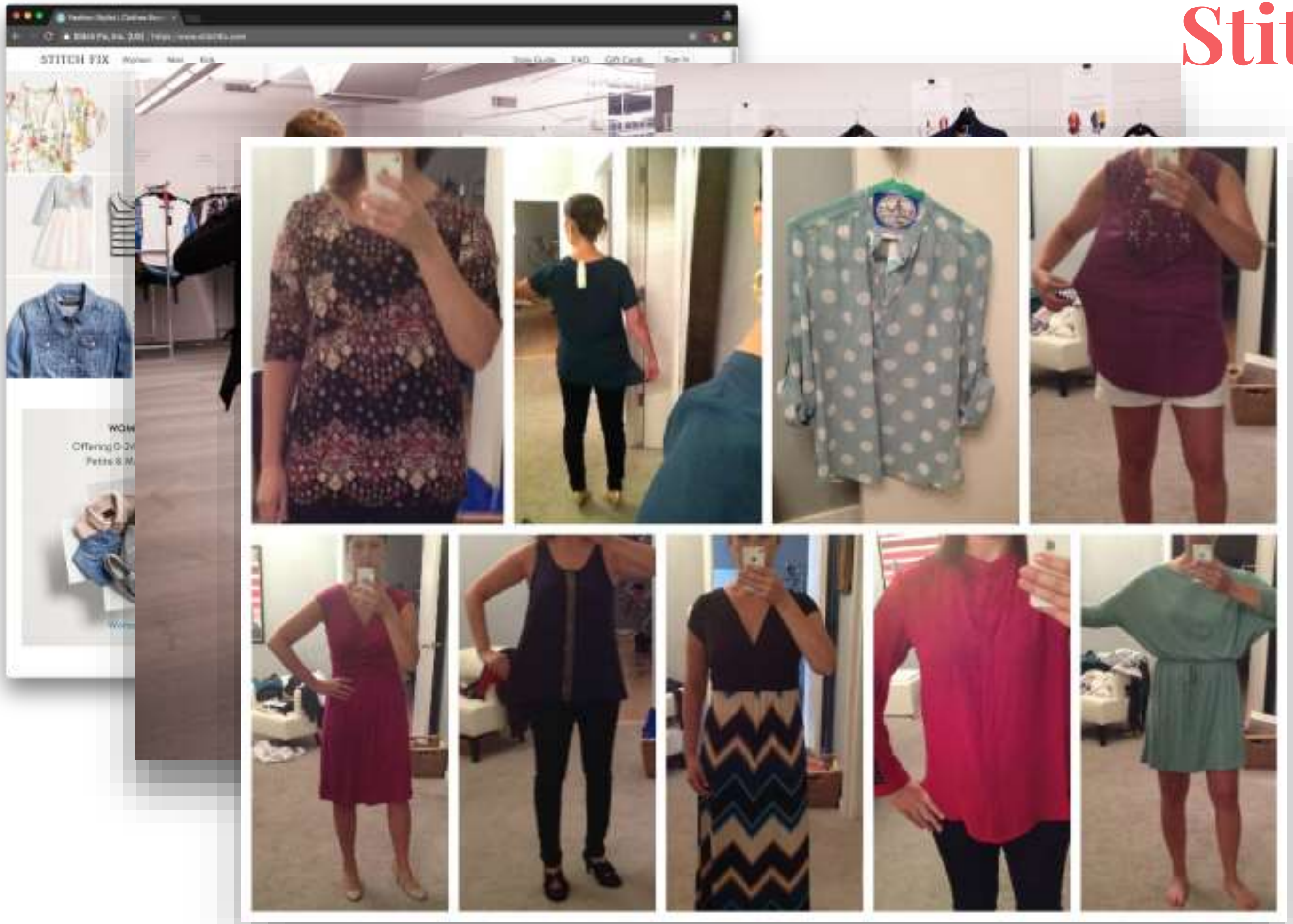


STITCH FIX
WOMEN
9:41 AM
MY FIX
Your Fix is scheduled to arrive by:
MAR 03
Saturday
2018
Introducing Extras – easily add intimates to your Fix, including camis, shapewear and tights.
View Extras
TELL US ABOUT YOUR STYLE
Would you wear this?

Stitch Fix



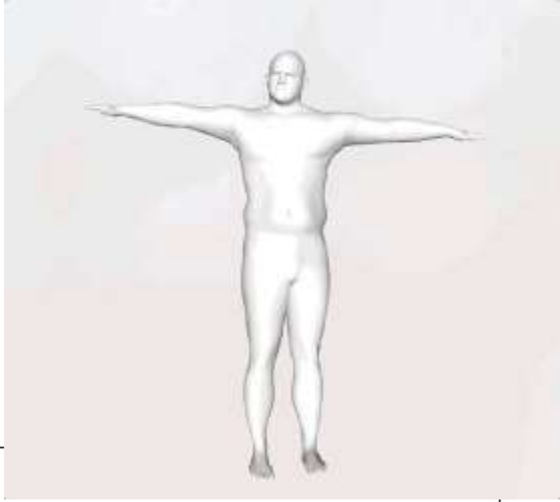
Stitch Fix

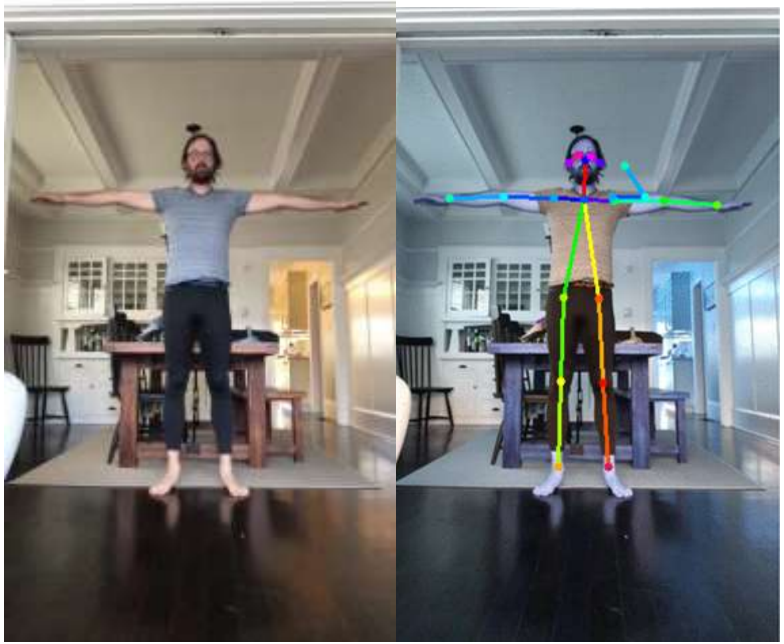


Stitch Fix

AI at Stitch Fix

If you're interested, ask me about this later!

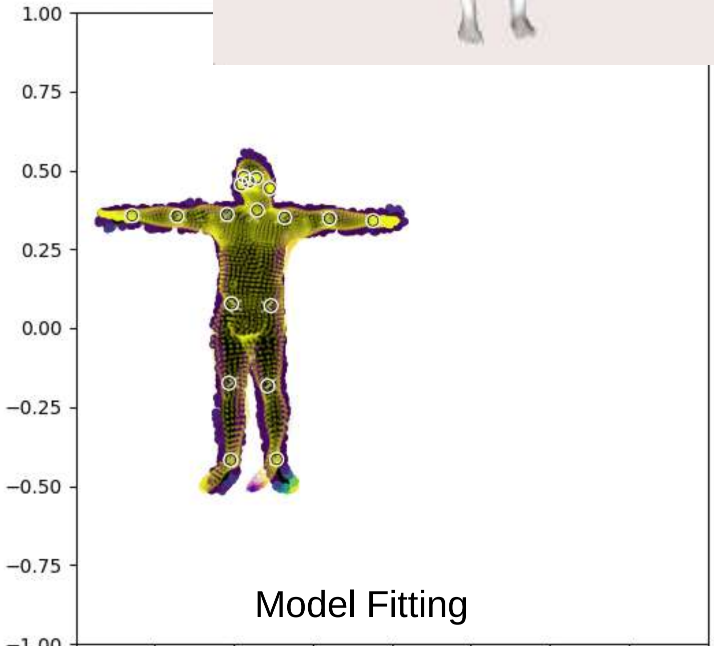




Capture

Joint Annotation

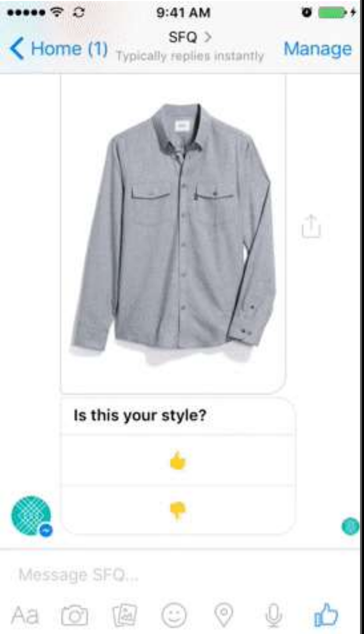
Segmentation

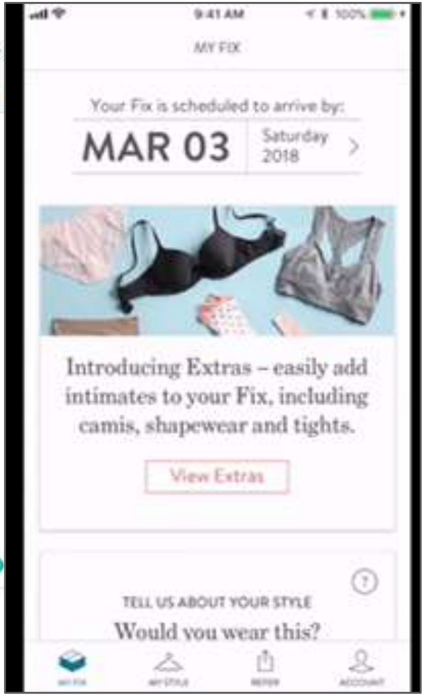


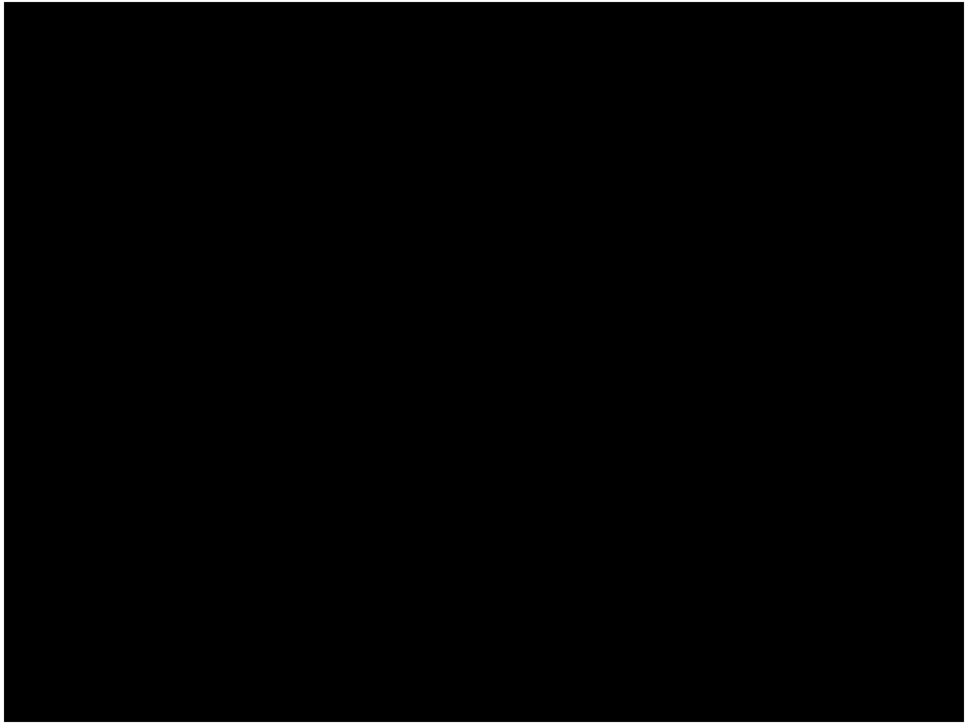
Model Fitting

"Latent" Style Space

Matrix Factorization

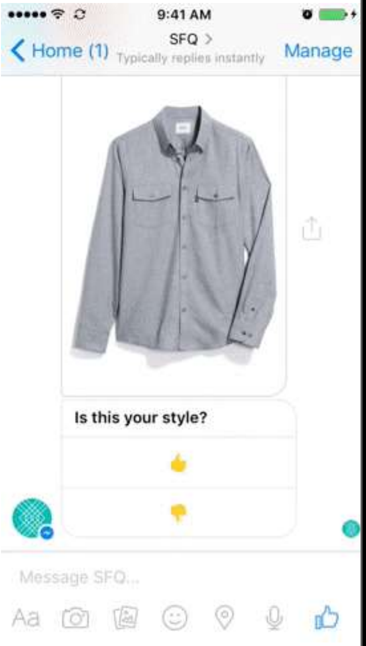


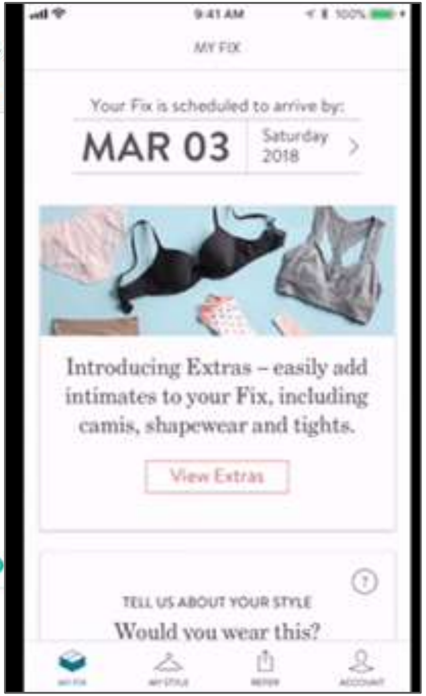


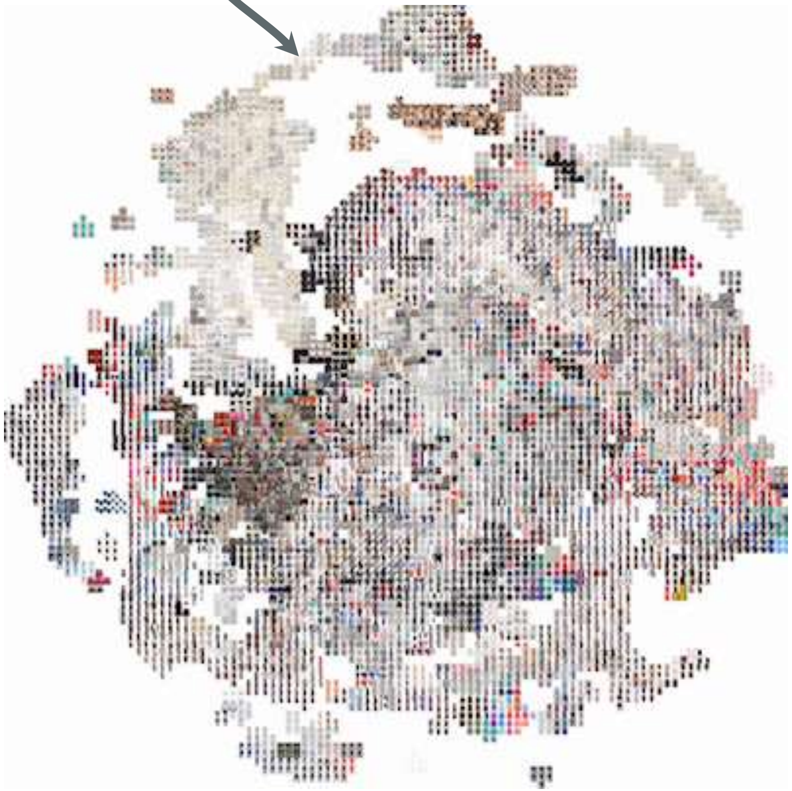


“Latent” Style Space

Matrix
Factorization

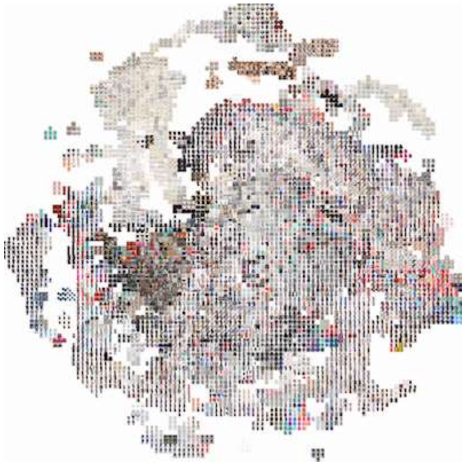






t-SNE Viz

100+ Data Scientists



Increased Personalization

Increased Personalization
Decreased Client Churn

Increased Personalization
Decreased Client Churn
Increased Item Sales

Increased Personalization
Decreased Client Churn
Increased Item Sales
Better Merch Buying

Increased Personalization
Decreased Client Churn
Increased Item Sales
Better Merch Buying
Better Stylist Relationships

Lessons Learned

1. **More data** means more personalization

1. Recommendation engines are **instruments** of your business.

1. **Custom** models respect your heterogeneous and unique system

~~Lessons Learned~~ Goals for Today

1. **More data** means more personalization

We'll use rec engines to drive personalization

1. Recommendation engines are **instruments** of your business.

The latent factors they reveal enable new science!

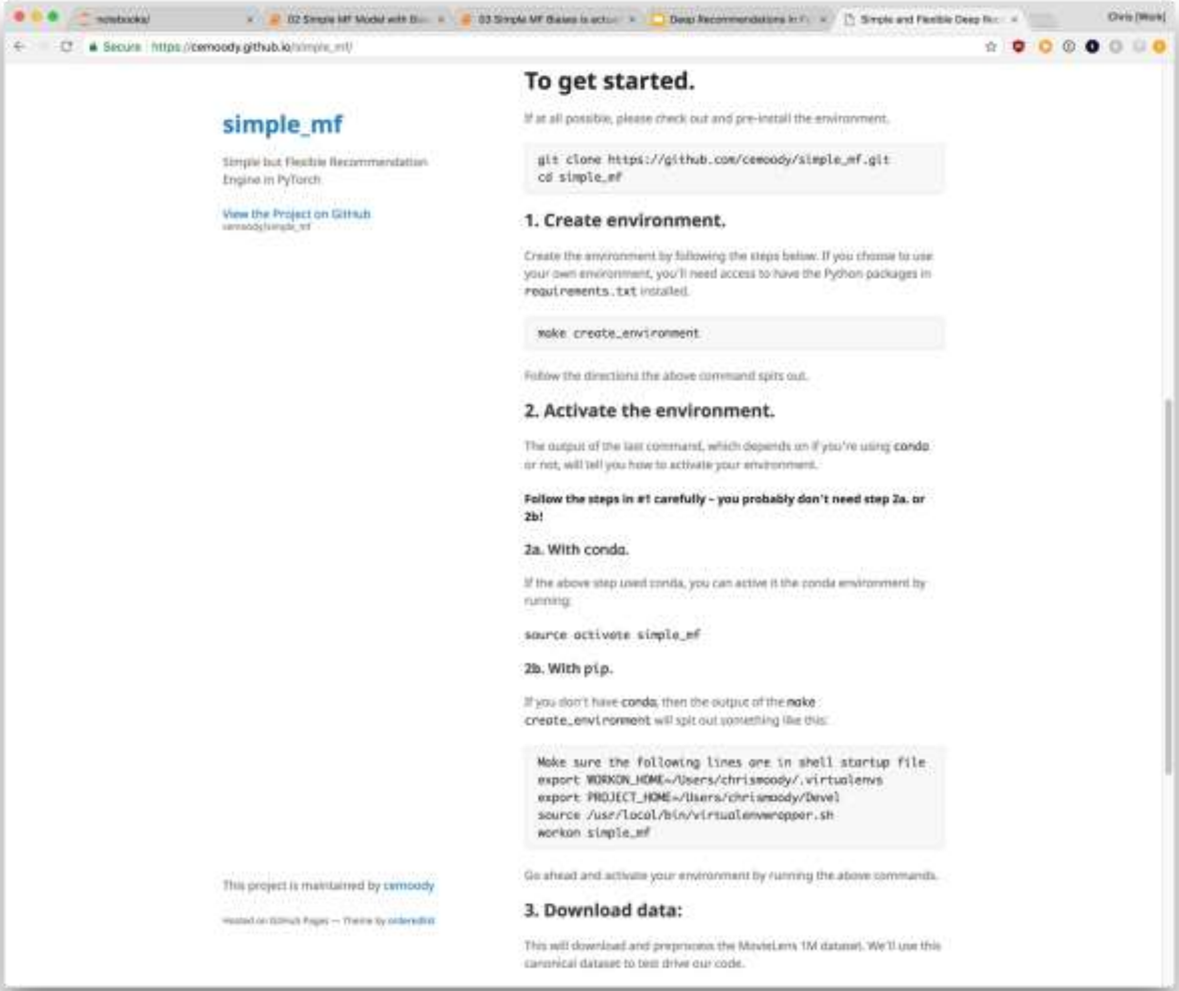
1. **Custom** models respect your heterogeneous and unique system

We'll explore 8 different ways of building deep recommendation engines.

Setup

Follow along with instructions here:

cemoody.github.io/simple_mf



Why PyTorch

functional-ish

declarative-ish

function

```
# Neural net architecture
x = chainer.Variable(x_data, volatile=not train)
t = chainer.Variable(y_data, volatile=not train)
h0 = model.embed(x)
h1_in = model.l1_x(F.dropout(h0, train=train))
          + model.l1_h(state['h1'])
c1, h1 = F.lstm(state['c1'], h1_in)
h2_in = model.l2_x(F.dropout(h1, train=train))
          + model.l2_h(state['h2'])
c2, h2 = F.lstm(state['c2'], h2_in)
y = model.l3(F.dropout(h2, train=train))
state = {'c1': c1, 'h1': h1, 'c2': c2, 'h2': h2}
loss = F.softmax_cross_entropy(y, t)
```

PYTORCH

```
model = Sequential()
model.add(Embedding(max_features, 128))
# try using a GRU instead, for fun
model.add(LSTM(128, 128))
model.add(Dropout(0.5))
model.add(Dense(128, 1))
model.add(Activation('sigmoid'))

# try using different optimizers
# and different optimizer configs
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              class_mode="binary")

print("Train...")
model.fit(X_train, y_train,
          batch_size=batch_size,
          nb_epoch=4,
          validation_data=(X_test, y_test),
          show_accuracy=True)
score, acc = model.evaluate(X_test, y_test,
                            batch_size=batch_size,
                            show_accuracy=True)
```

K Keras

functional-ish

declarative-ish

compile

```
# Neural net architecture
x = chainer.Variable(x_data, volatile=not train)
t = chainer.Variable(y_data, volatile=not train)
h0 = model.embed(x)
h1_in = model.l1_x(F.dropout(h0, train=train))
            + model.l1_h(state['h1'])
c1, h1 = F.lstm(state['c1'], h1_in)
h2_in = model.l2_x(F.dropout(h1, train=train))
            + model.l2_h(state['h2'])
c2, h2 = F.lstm(state['c2'], h2_in)
y = model.l3(F.dropout(h2, train=train))
state = {'c1': c1, 'h1': h1, 'c2': c2, 'h2': h2}
loss = F.softmax_cross_entropy(y, t)
```

```
model = Sequential()
model.add(Embedding(max_features, 128))
# try using a GRU instead, for fun
model.add(LSTM(128, 128))
model.add(Dropout(0.5))
model.add(Dense(128, 1))
model.add(Activation('sigmoid'))

# try using different optimizers
# and different optimizer configs
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              class_mode="binary")

print("Train...")
model.fit(X_train, y_train,
          batch_size=batch_size,
          nb_epoch=4,
          validation_data=(X_test, y_test),
          show_accuracy=True)
score, acc = model.evaluate(X_test, y_test,
                            batch_size=batch_size,
                            show_accuracy=True)
```

PYTORCH

Keras

functional-ish

```
# Neural net architecture
x = chainer.Variable(x_data, volatile=not train)
t = chainer.Variable(y_data, volatile=not train)
h0 = model.embed(x)
h1_in = model.l1_x(F.dropout(h0, train=train))
                 + model.l1_h(state['h1'])
c1, h1 = F.lstm(state['c1'], h1_in)
h2_in = model.l2_x(F.dropout(h1, train=train))
                 + model.l2_h(state['h2'])
c2, h2 = F.lstm(state['c2'], h2_in)
y = model.l3(F.dropout(h2, train=train))
state = {'c1': c1, 'h1': h1, 'c2': c2, 'h2': h2}
loss = F.softmax_cross_entropy(y, t)
```

data

PYTORCH

declarative-ish

```
model = Sequential()
model.add(Embedding(max_features, 128))
# try using a GRU instead, for fun
model.add(LSTM(128, 128))
model.add(Dropout(0.5))
model.add(Dense(128, 1))
model.add(Activation('sigmoid'))

# try using different optimizers
# and different optimizer configs
model.compile(loss='binary_crossentropy',
              optimizer='adam',
              class_mode="binary")

print("Train...")
model.fit(X_train, y_train,
          batch_size=batch_size,
          nb_epoch=4,
          validation_data=(X_test, y_test),
          show_accuracy=True)
score, acc = model.evaluate(X_test, y_test,
                            batch_size=batch_size,
                            show_accuracy=True)
```

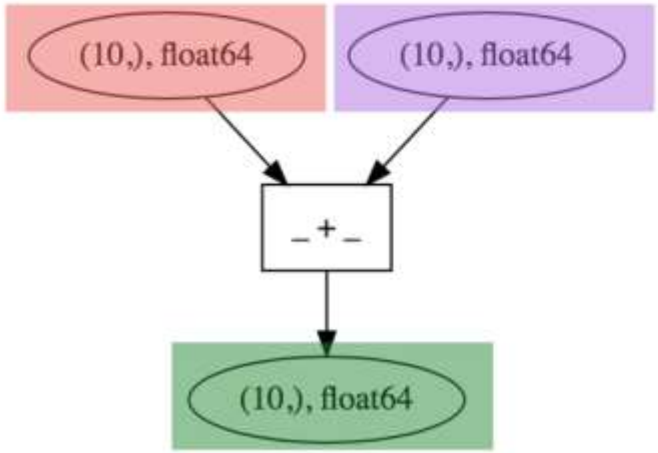
K Keras

The low level

```
x = Variable(np.ones(10))
y = Variable(np.ones(10))
loss = x + y
```

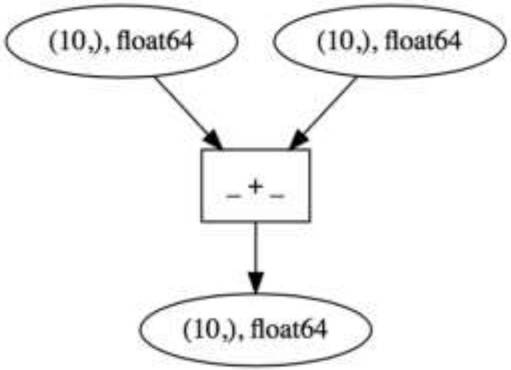
The low level

```
x = Variable(np.ones(10))
y = Variable(np.ones(10))
loss = x + y
```



TensorFlow theano

```
x = t.vector('x')
y = t.vector('y')
loss = x + y
```

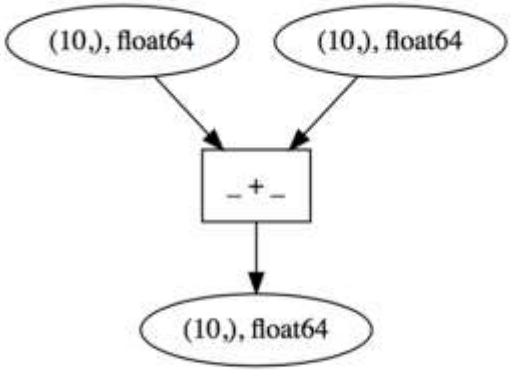



```
In [47]: loss
Out[47]: theano.tensor.var.TensorVariable
```

symbolic variable

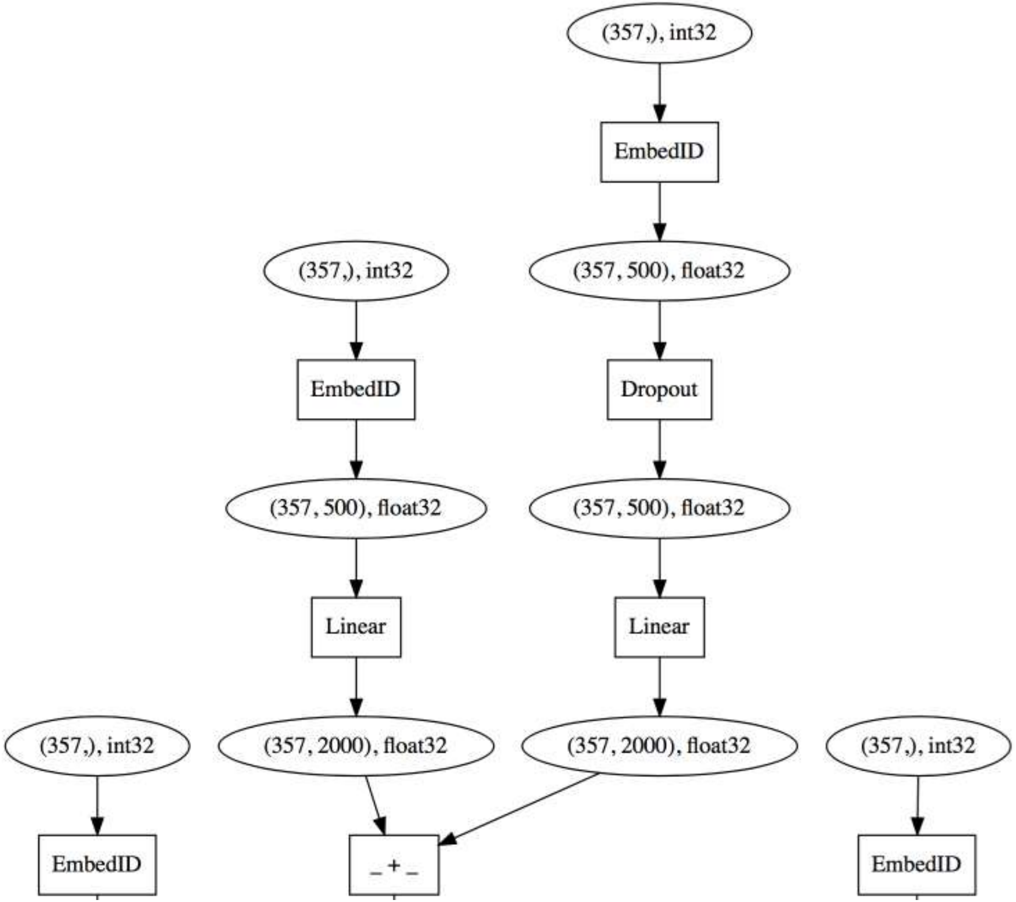
PYTORCH

```
x = Variable(np.ones(10))
y = Variable(np.ones(10))
loss = x + y
```



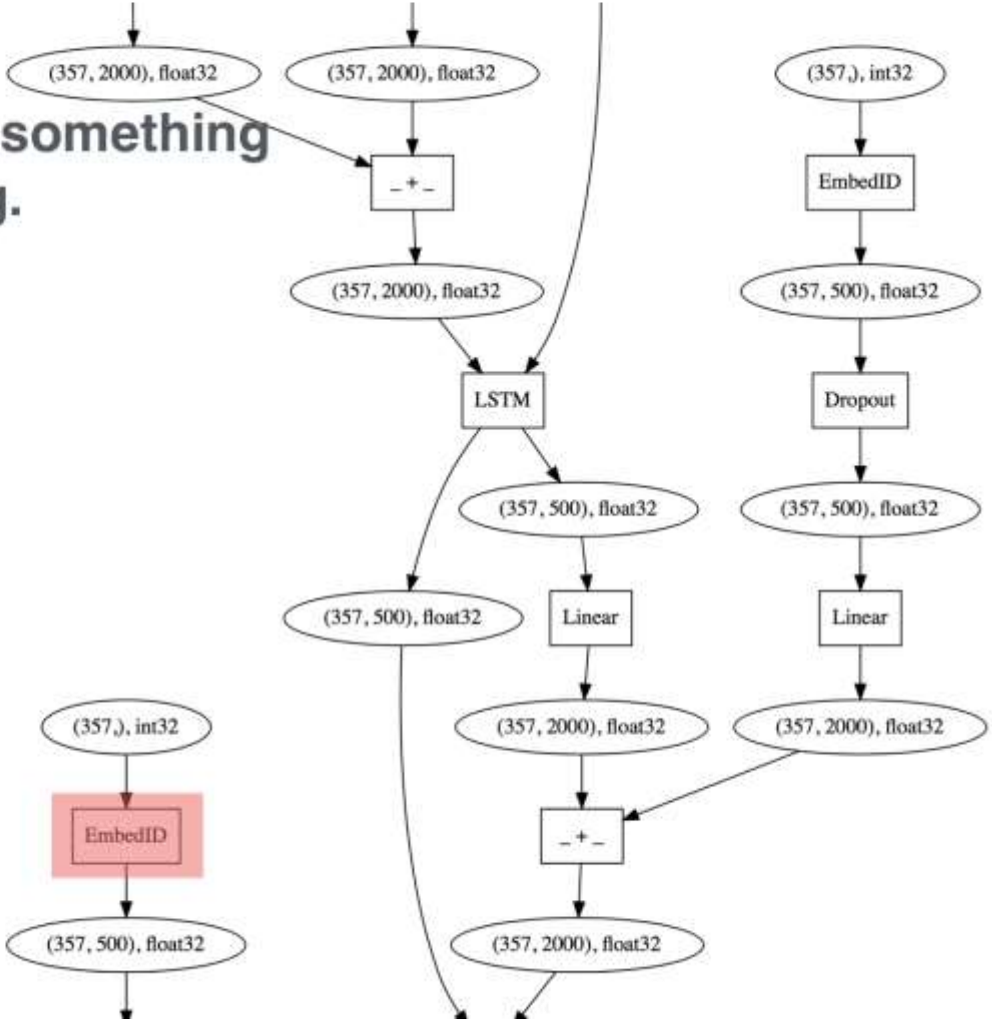

```
In [47]: loss.data
Out[47]: array([ 2.,  2.,  2.,  2.,  2., 2.]
```

symbolic + numeric variable



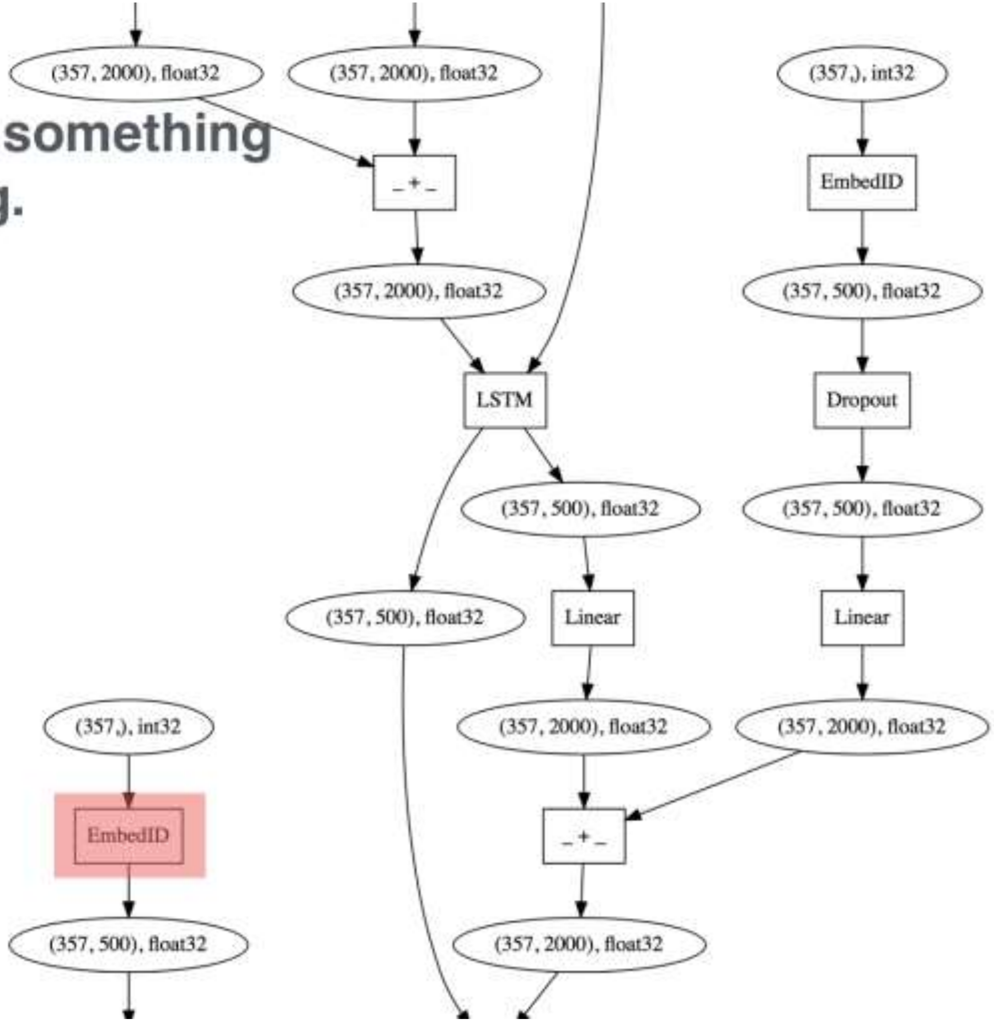
This gets *very* deep.

…and then something goes wrong.



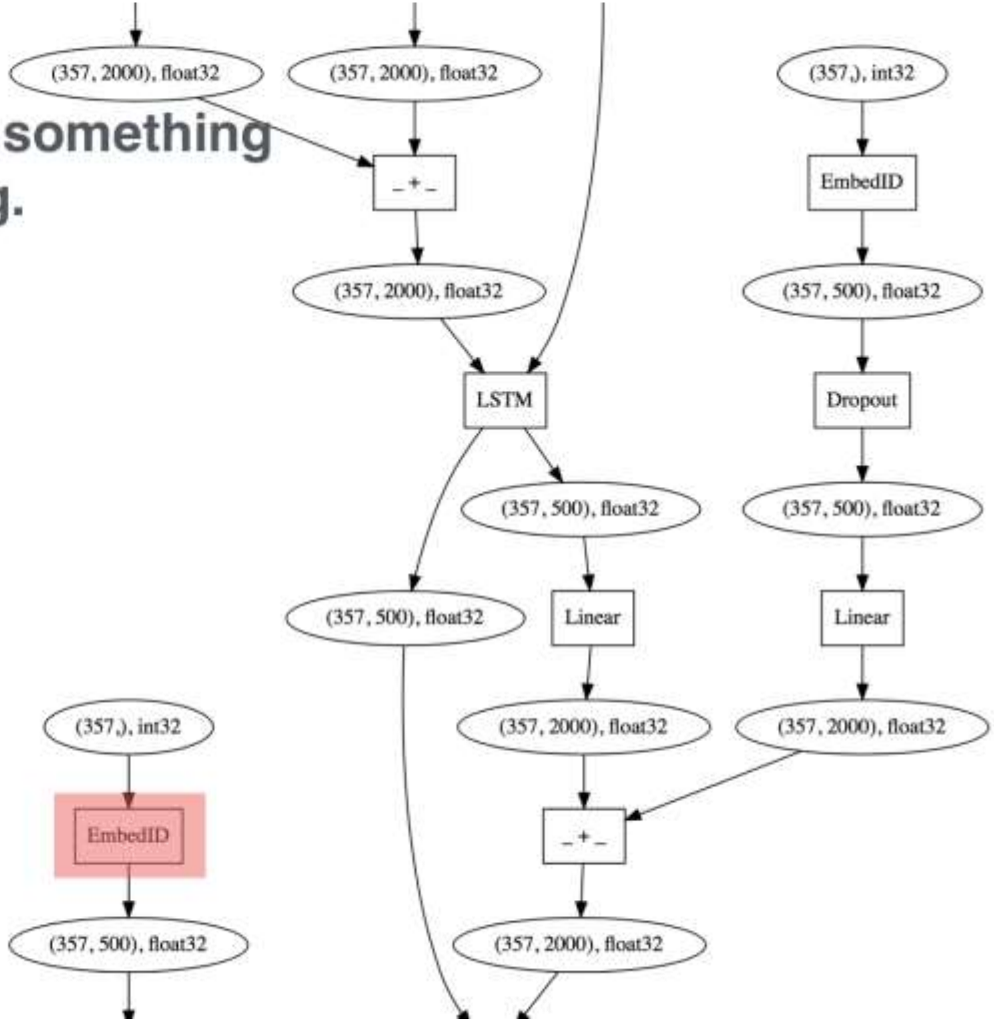
…PyTorch computes everything at run time… so debug & investigate!

…and then something goes wrong.



…PyTorch computes everything at run time… so debug & investigate!

…and then something goes wrong.



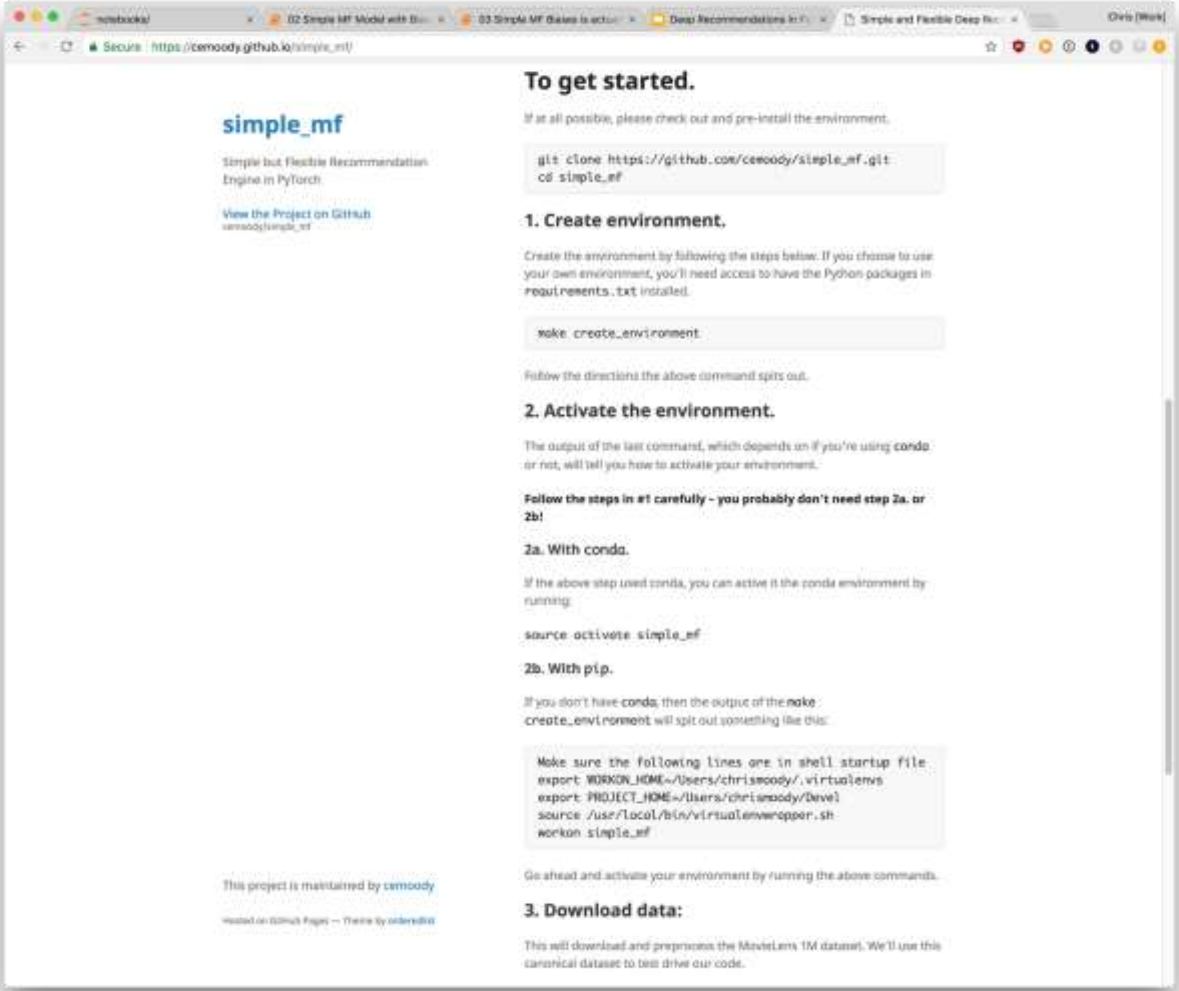

```
In [47]: z.data
Out[47]: array([ 2.,  2.,  2.,  nan,  2., 2.])
```

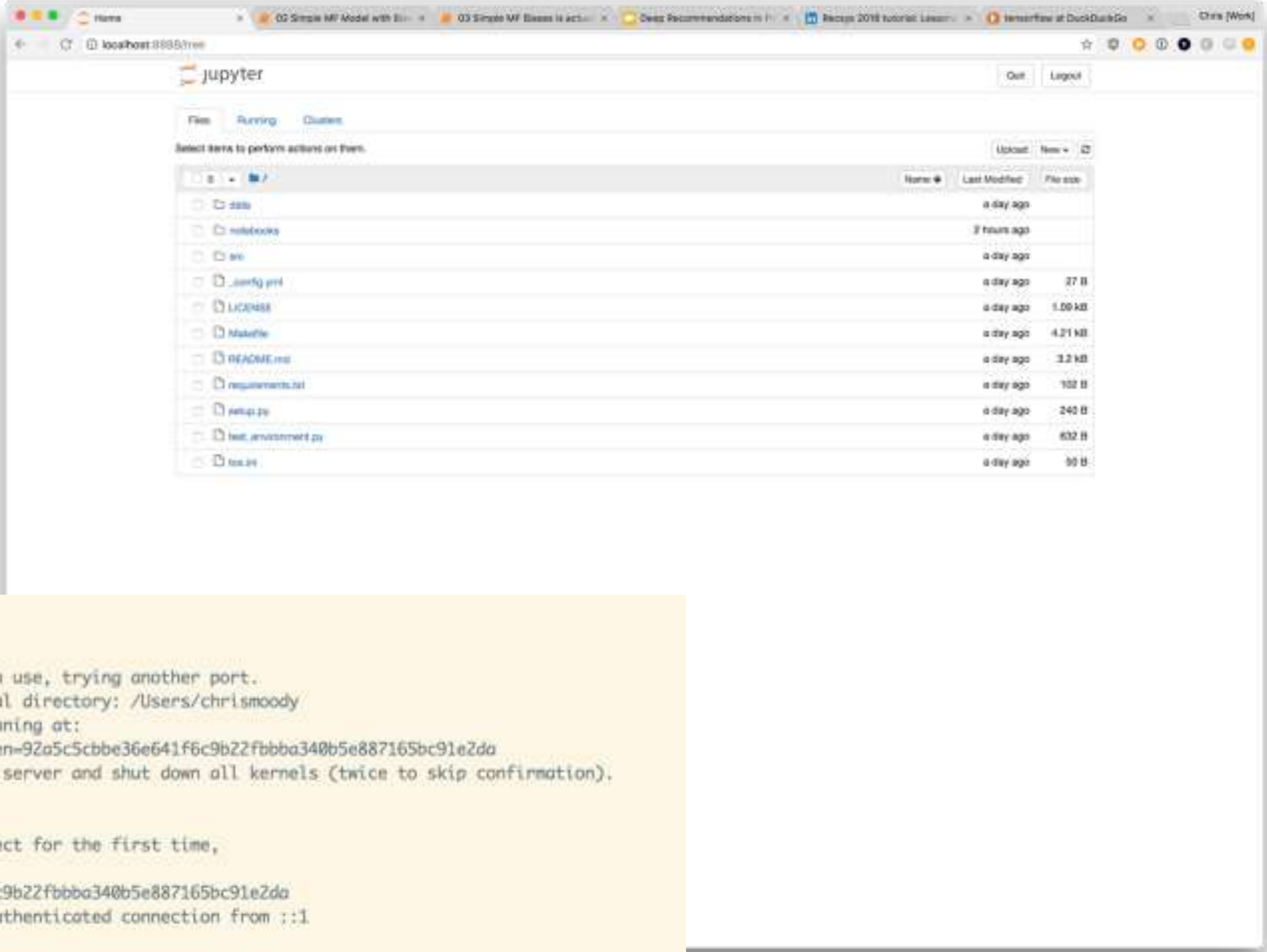
Setup

Setup

Follow along with instructions here:

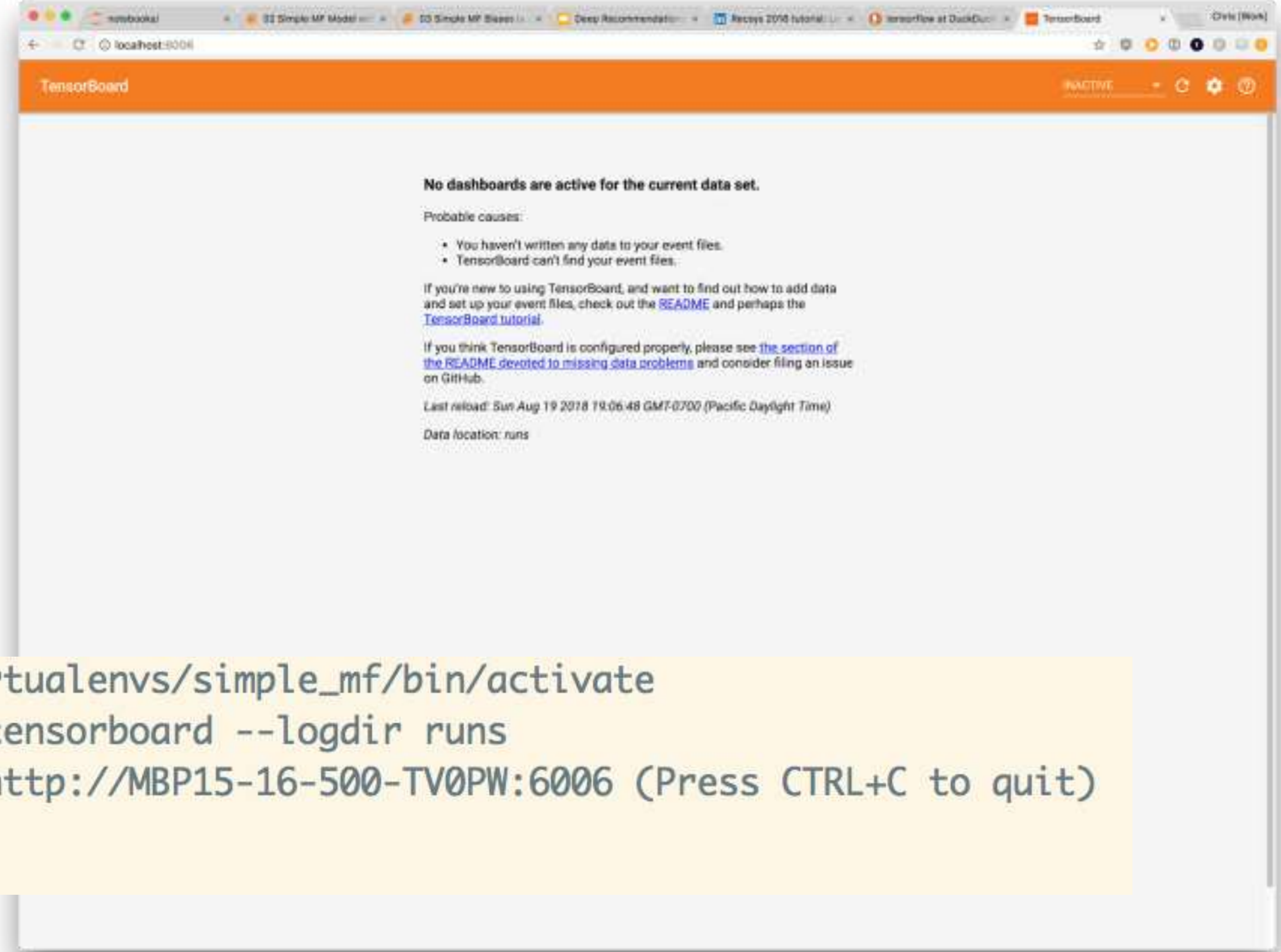
cemoody.github.io/simple_mf





Start Jupyter Notebook

Start Tensorboard



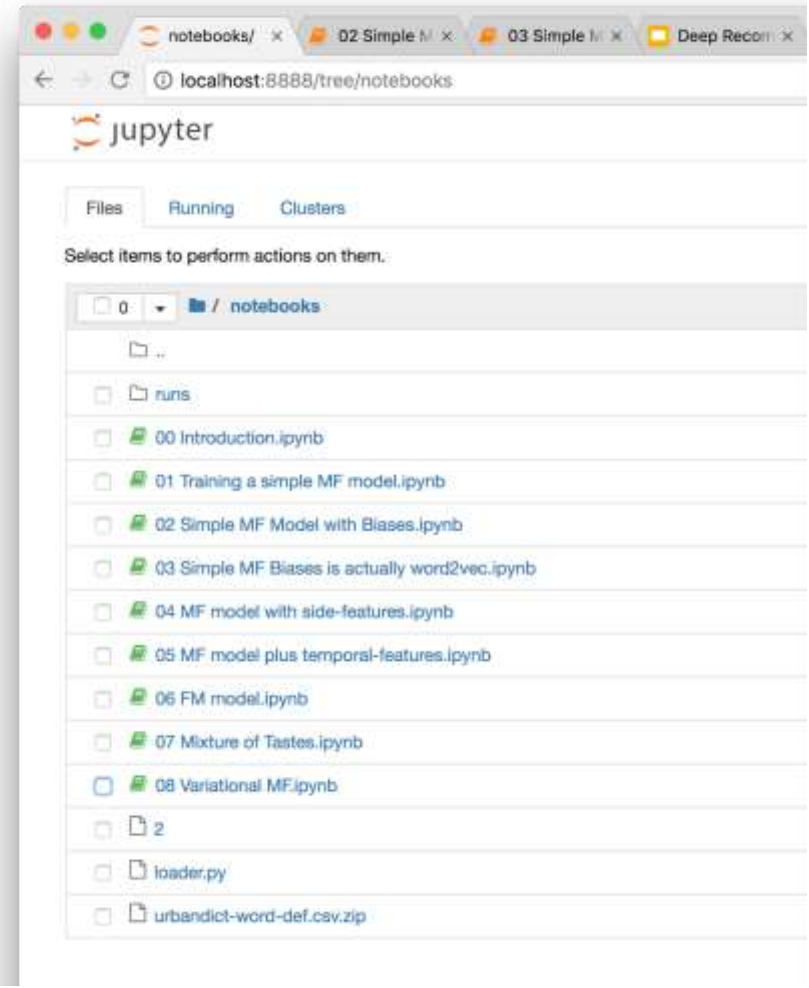

```
bash-3.2$ source ~/.virtualenvs/simple_mf/bin/activate
(simple_mf) bash-3.2$ tensorboard --logdir runs
TensorBoard 1.10.0 at http://MBP15-16-500-TV0PW:6006 (Press CTRL+C to quit)
```

There are eight Jupyter Notebooks in this tutorial.

We'll go through them one-by-one.

Feel free to run them in advance -- they'll start training models in the background.

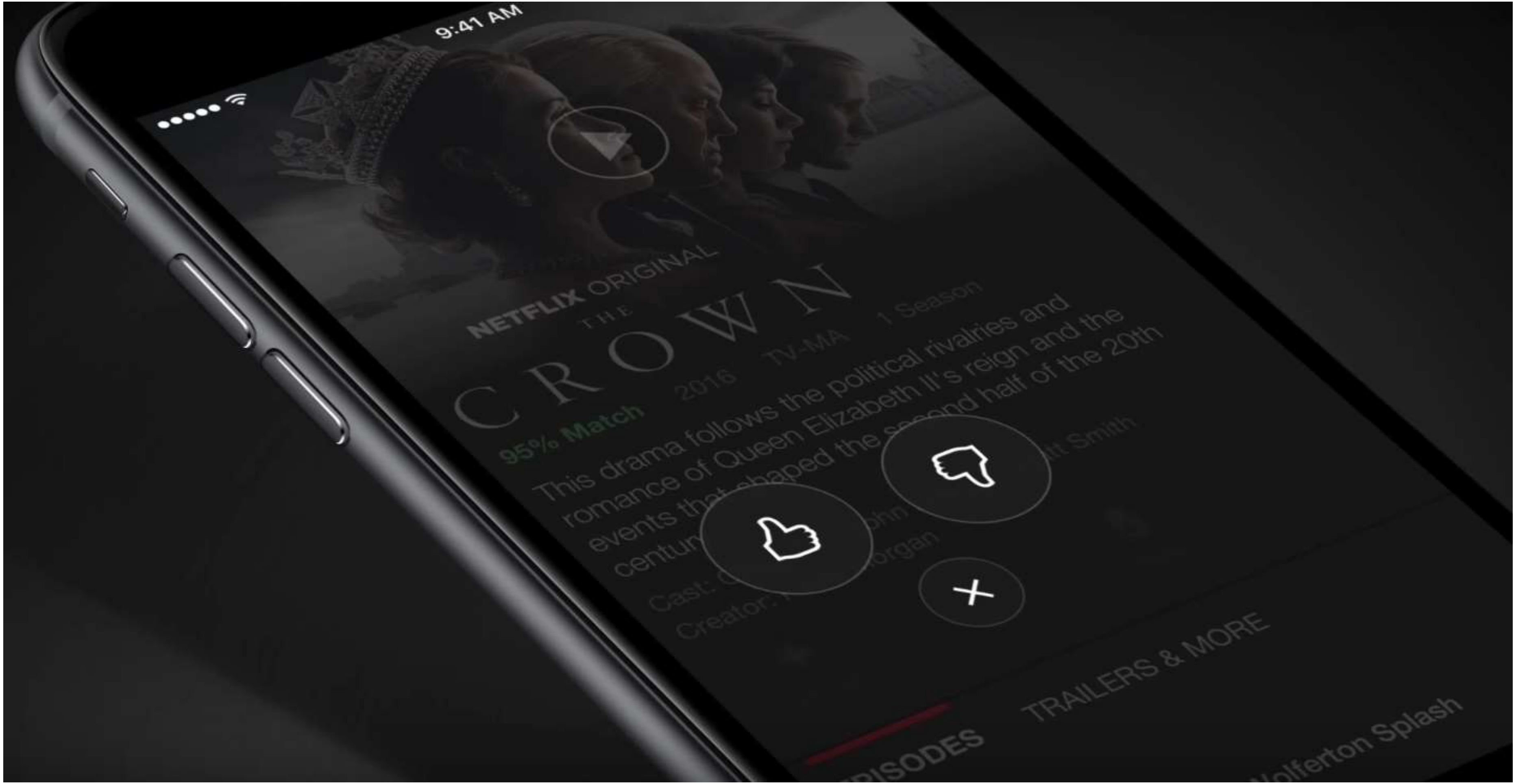
… or start them up as I go along.



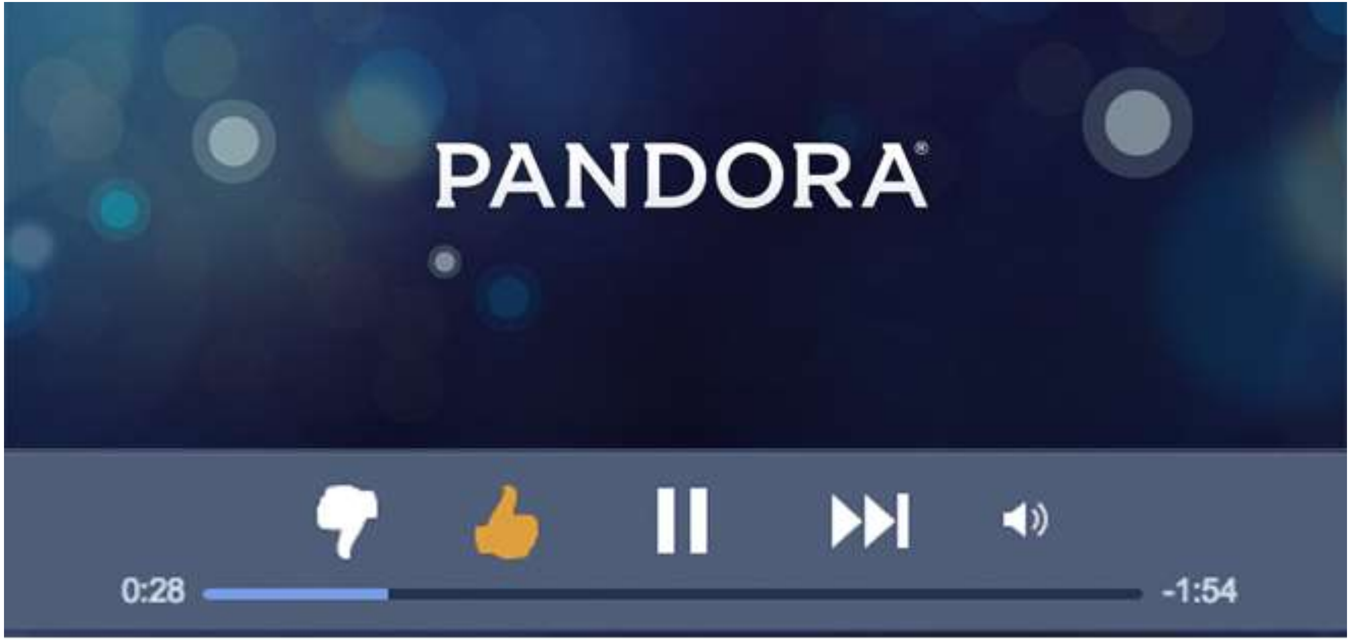
Matrix Factorization Fundamentals

Notebooks we'll be using:

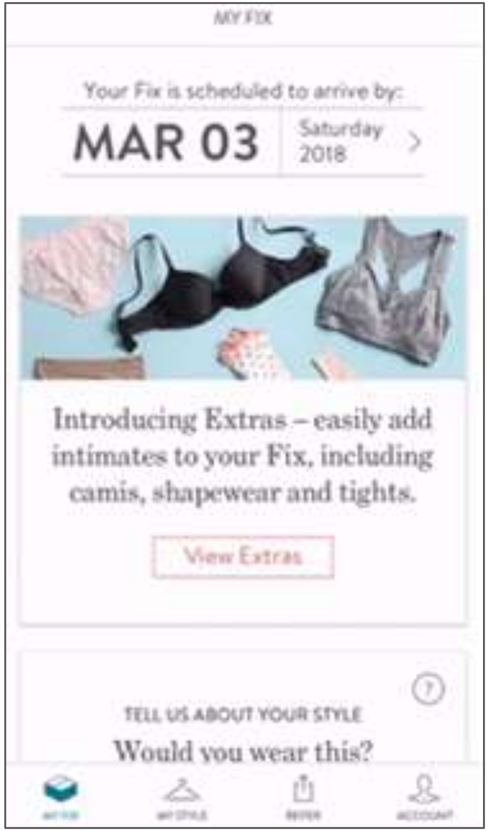
01 Training a simple MF model.ipynb
02 Simple MF Model with Biases.ipynb



9:41 AM
NETFLIX ORIGINAL
THE
CROWN
95% Match
2016
TV-MA
1 Season
This drama follows the political rivalries and
romance of Queen Elizabeth II's reign and the
TRAILERS & MORE

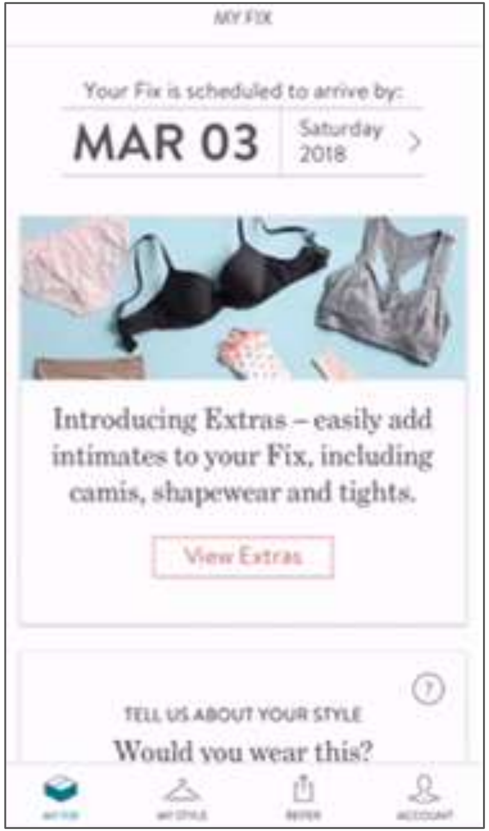

PANDORA
0:28
-1:54

+1


MY FIX
Your Fix is scheduled to arrive by:
MAR 03
Saturday
2018
Introducing Extras – easily add intimates to your Fix, including camis, shapewear and tights.
View Extras
TELL US ABOUT YOUR STYLE
Would you wear this?

+1

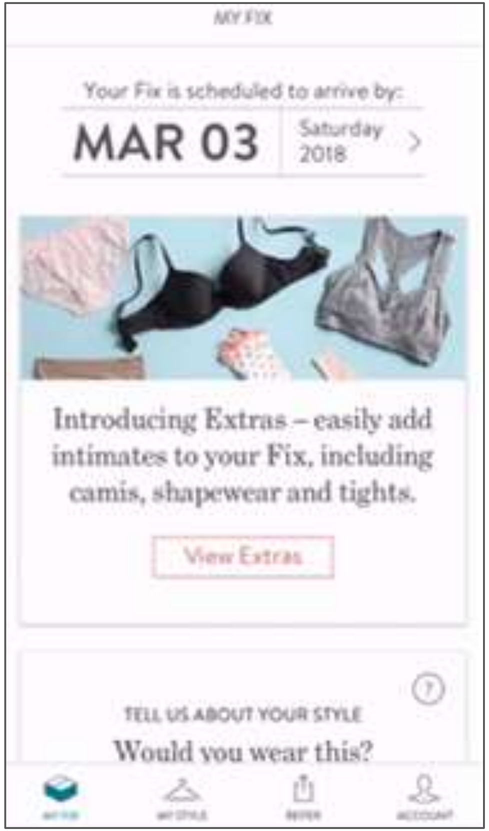
0


MY FIX
Your Fix is scheduled to arrive by:
MAR 03
Saturday
2018
Introducing Extras – easily add intimates to your Fix, including camis, shapewear and tights.
View Extras
TELL US ABOUT YOUR STYLE
Would you wear this?

+1

0

0

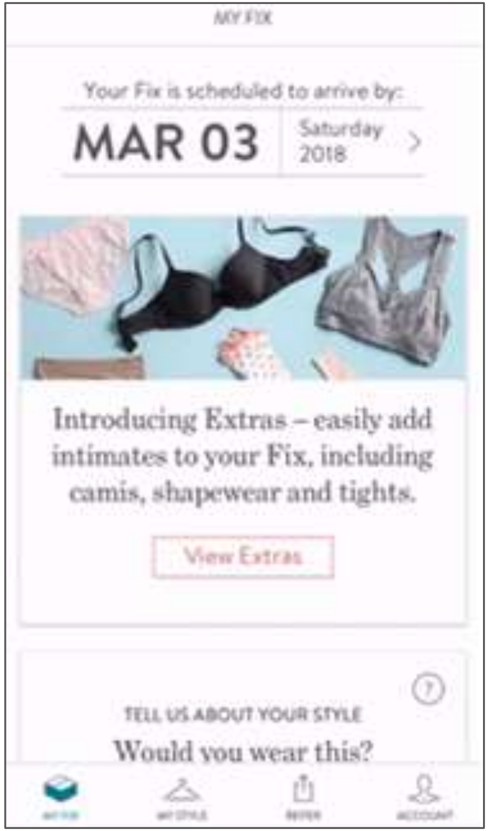

MY FIX
Your Fix is scheduled to arrive by:
MAR 03
Saturday
2018
Introducing Extras – easily add intimates to your Fix, including camis, shapewear and tights.
View Extras
TELL US ABOUT YOUR STYLE
Would you wear this?

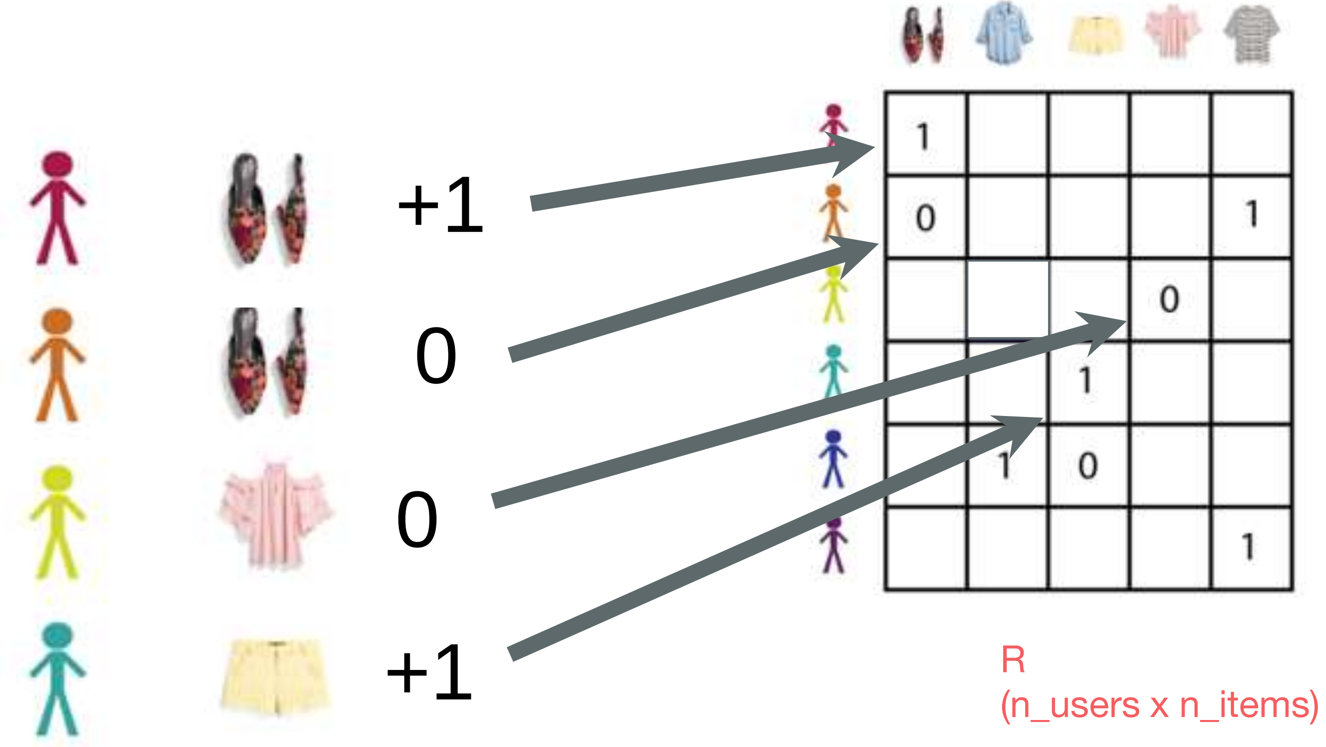
+1

0

0

+1


MY FIX
Your Fix is scheduled to arrive by:
MAR 03
Saturday
2018
Introducing Extras – easily add intimates to your Fix, including camis, shapewear and tights.
View Extras
TELL US ABOUT YOUR STYLE
Would you wear this?


+1
0
0
+1
1
0
1
0
1
1
0
1
R
(n_users x n_items)

R

The ***ratings matrix***
(n_users x n_items)

1				
0				1
			0	
		1		
	1	0		
				1

Extremely large
(Easily tens of billions of elements)

Mostly zeroes
(Typical sparsity is 0.01% - 1%)

R

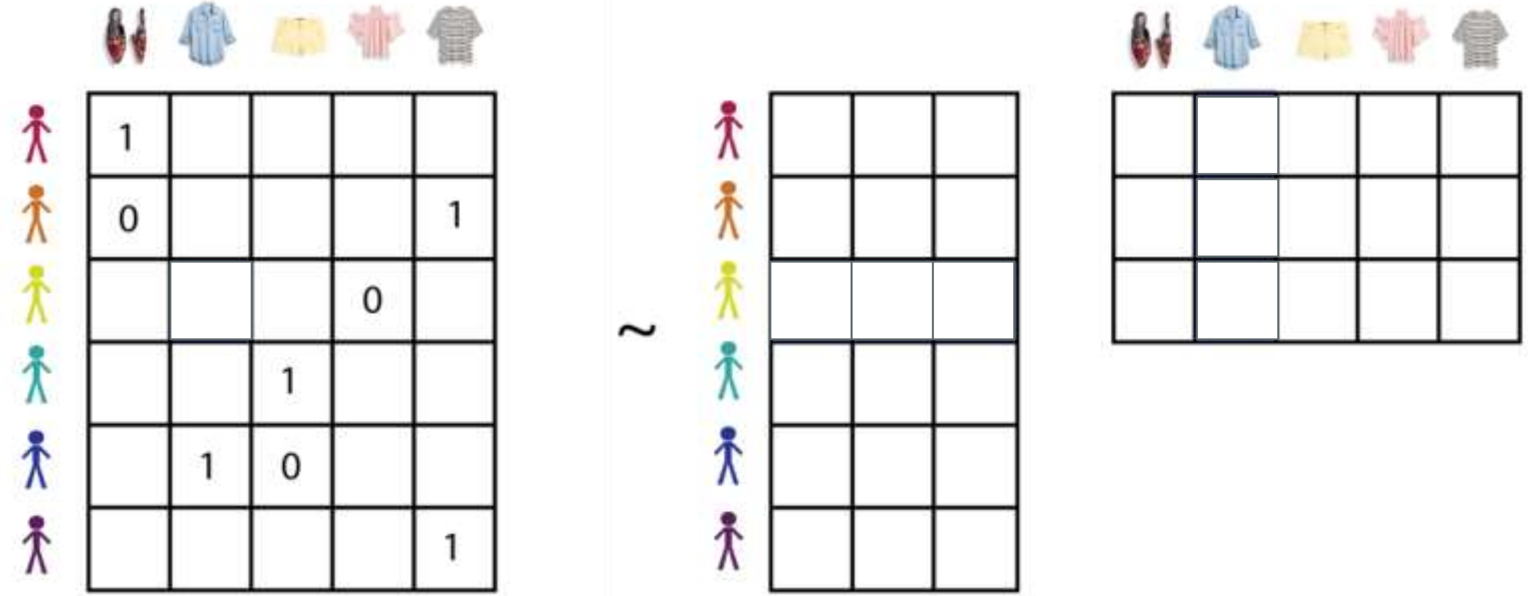
The ***ratings matrix***
(n_users, n_items)

P

The ***user matrix***
(n_users, k)

Q

The ***item matrix***
(n_item, k)



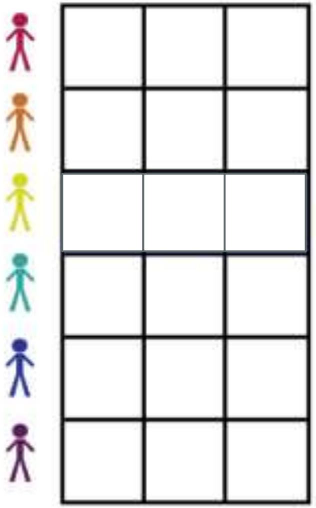
1				
0				1
			0	
		1		
	1	0		
				1

Much smaller!
(millions or 100ks of rows)

Compact & dense!
No zeroes, efficient storage.

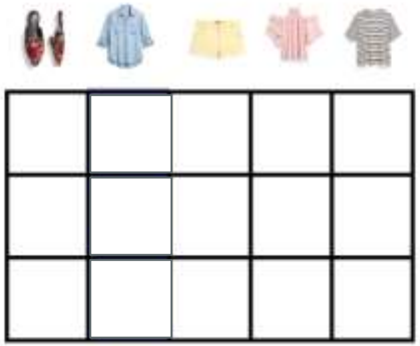
P

The ***user matrix***
(n_users, k)

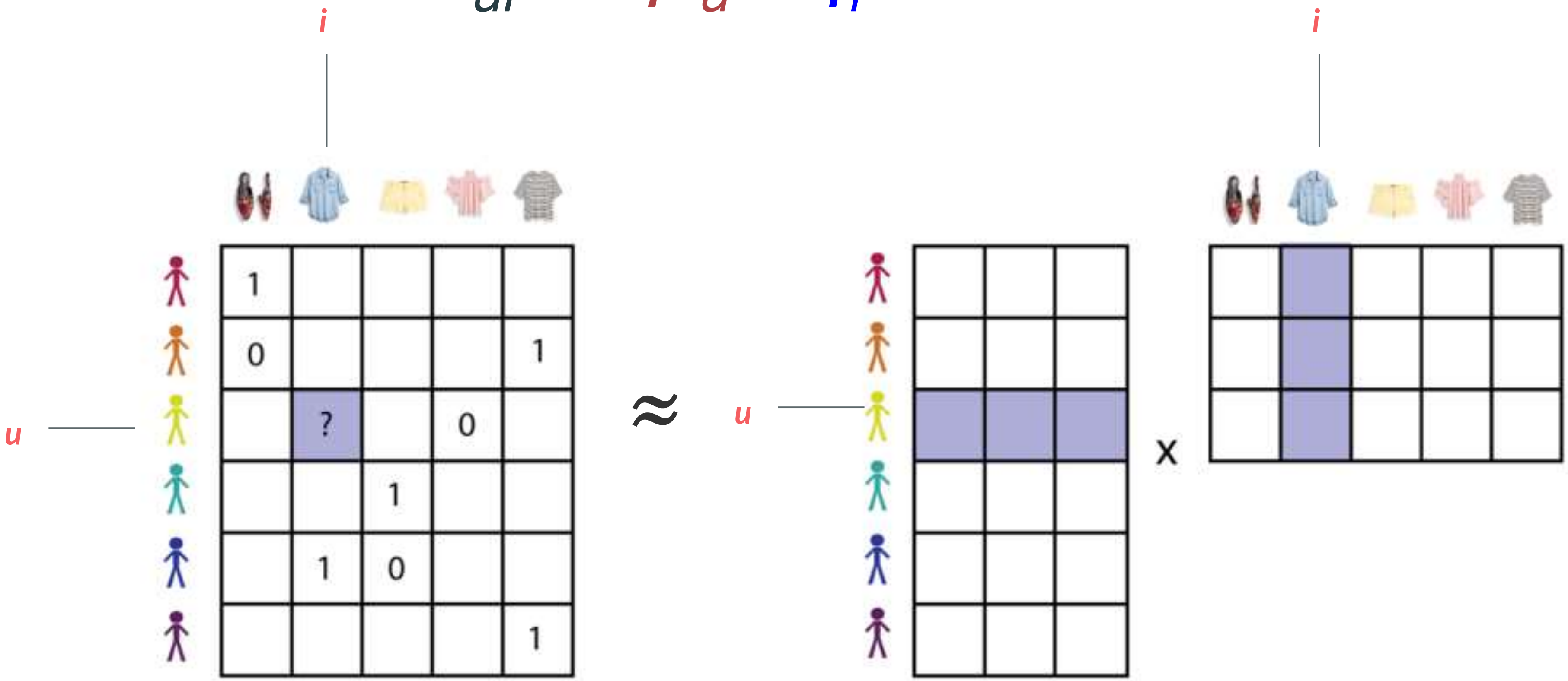


Q

The ***item matrix***
(n_item, k)



$$R_{ui} \approx p_u \cdot q_i$$

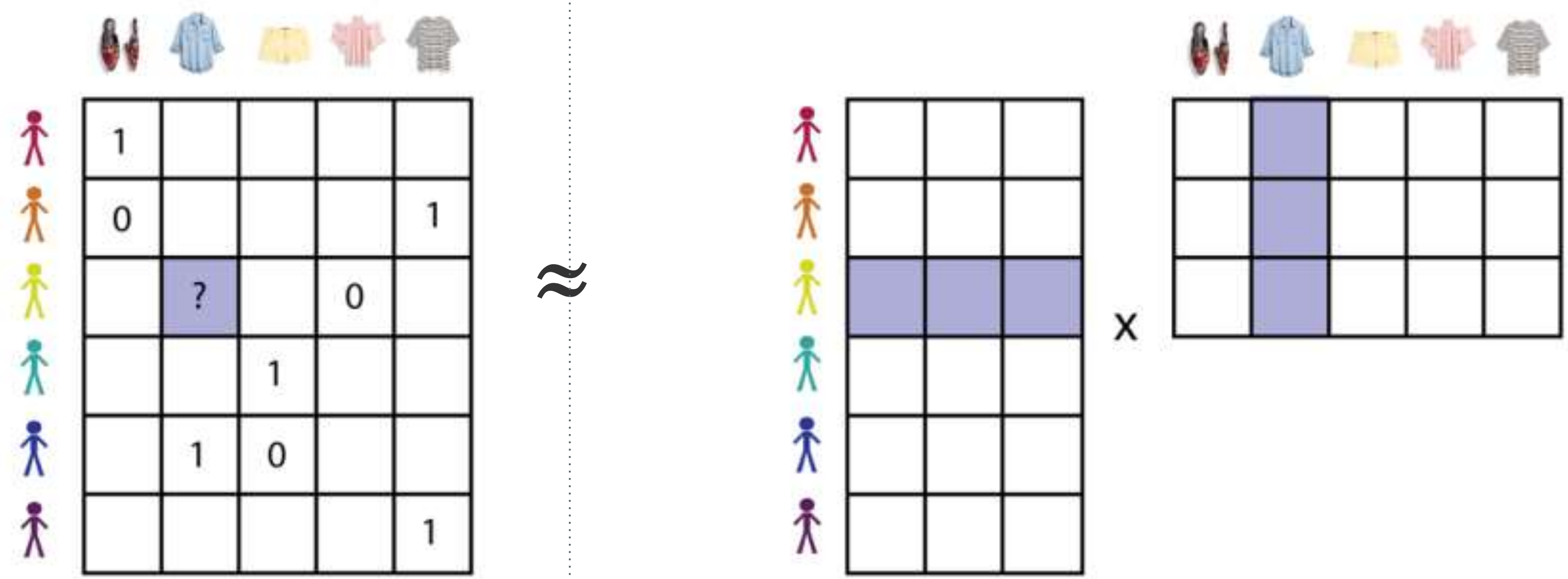


$$R_{ui} \approx p_u \cdot q_i$$

A single +1 or 0 rating

User vector

Item vector



known

to be estimated

$$\text{Minimize } L = (R_{ui} - p_u \cdot q_i)^2$$

$$Minimize\ L = (R_{ui} - p_u \cdot q_i)^2 + c\,|Q|^2 + c|P|^2$$

$$Minimize\ L = (R_{ui} - p_u \cdot q_i)^2 + c\,|Q|^2 + c|P|^2$$

Let's try it!

01 Training a simple MF model

$$R_{ui} = p_u \cdot q_i$$

Baseline model

- *Only* captures interactions.
- What if a user generally likes everything?
- What if an item is generally popular?

$$R_{ui} = p_u \cdot q_i$$

Baseline model

- *Only* captures interactions.
- What if a user generally likes everything?
- What if an item is generally popular?

Let's add *biases*.

$$R_{ui} = \boxed{b + \omega_u + \mu_i} + p_u \cdot q_i$$

Model +
global bias
user biases
item biases

Now we learn how much a user generally likes things, and an item is generally liked.

$$R_{ui} = b + \boxed{\omega_u + \mu_i} + p_u \cdot q_i$$

A single +1 or 0 rating

User bias

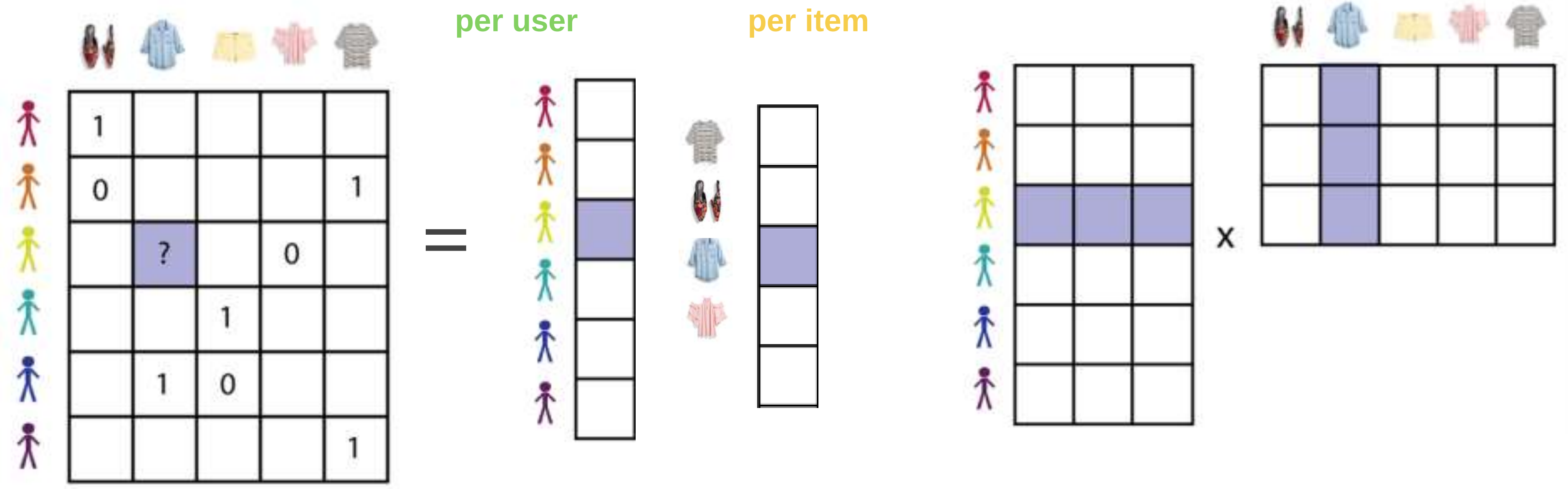
One number per user

Item bias

One number per item

User vector

Item vector



$$R_{ui} = b + \boxed{\omega_u + \mu_i} + p_u \cdot q_i$$

A single +1 or 0 rating

User bias

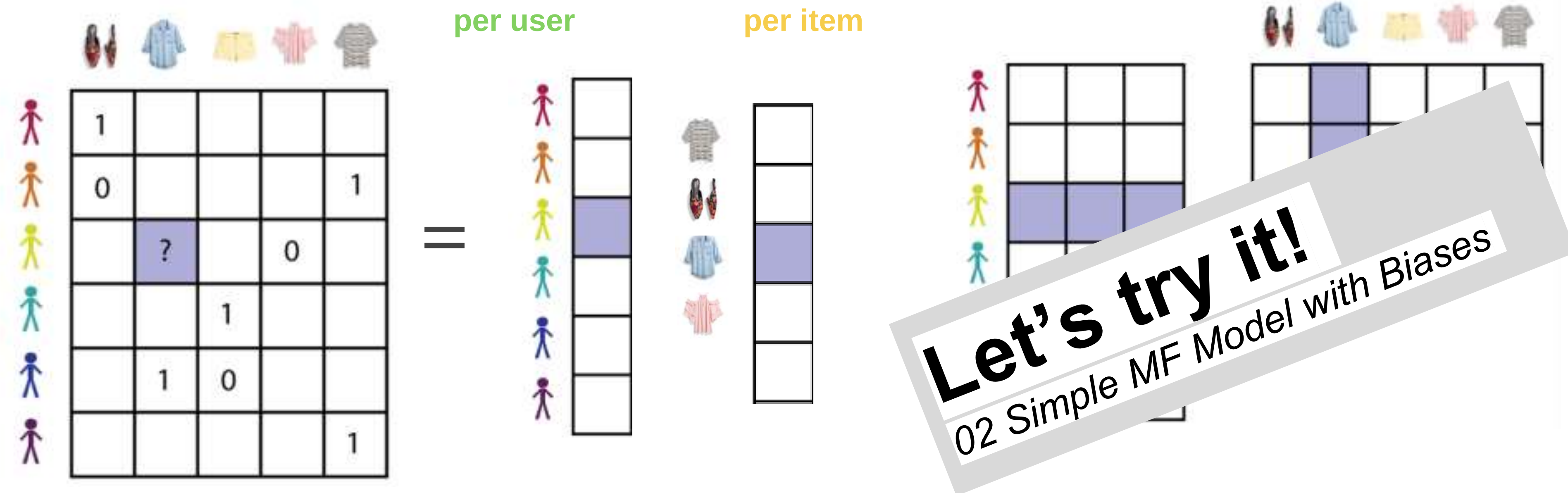
One number per user

Item bias

One number per item

User vector

Item vector



Let's try it!

02 Simple MF Model with Biases

Recommendation Engines
are an *instrument* to do science

User & item vectors are not black boxes.

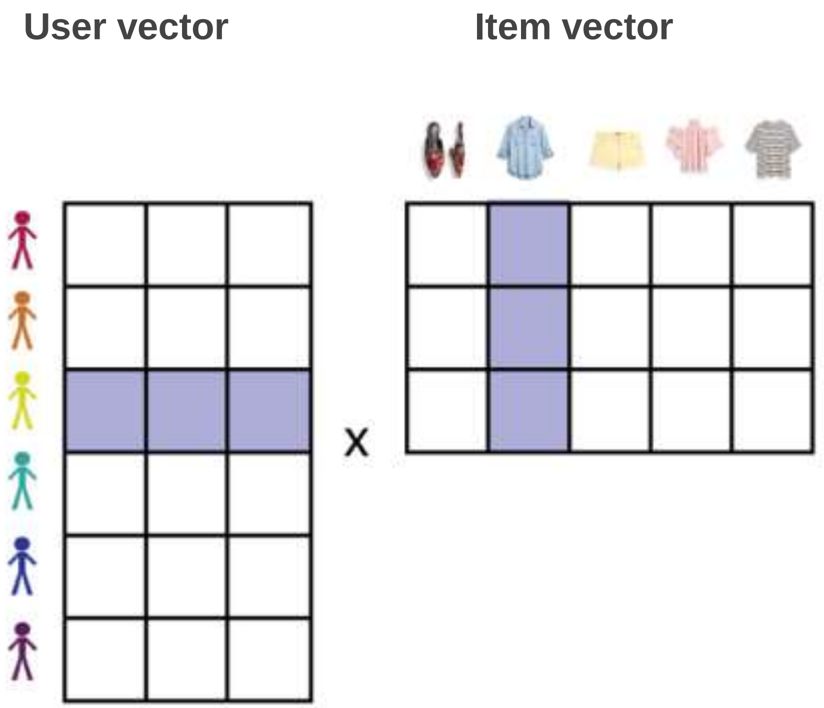
They are **instruments** into your space.

Stitch Fix vectors stand for **clothing style**.

Movielens vectors yield **movie genres**.

At Spotify, they represent latent **musical tastes**.

Amazon they represent latent **categories**.



Let’s PCA the Stitch Fix vectors and take a look.

+1.2	- 0.3	+0.4	-4.0	+0.9
+6.7	- 0.44	+ 0.9	+0.58	-0.7
+3.8	- 0.9	-2. 4	+0.8	+0.3

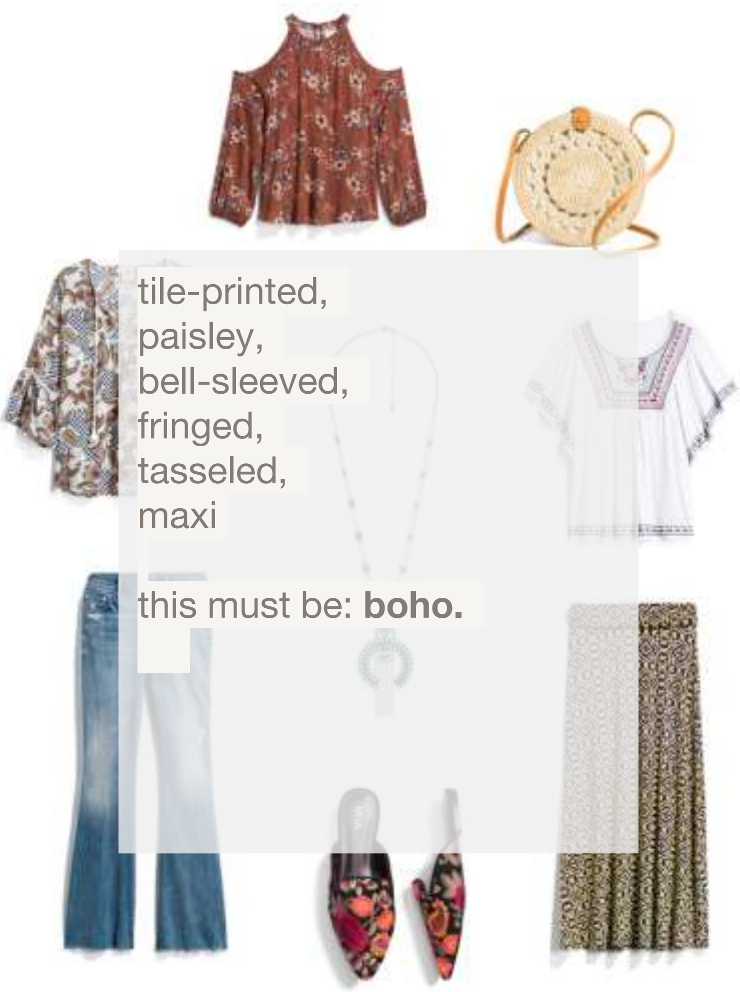
Let’s PCA the Stitch Fix vectors and take a look.

+1.2	- 0.3	+0.4	-4.0	+0.9
+6.7	- 0.44	+ 0.9	+0.58	-0.7
+3.8	- 0.9	-2. 4	+0.8	+0.3

We’ll sort by the first eigenvector.

Negative End

Positive End



tile-printed,
paisley,
bell-sleeved,
fringed,
tasseled,
maxi

this must be: **boho.**

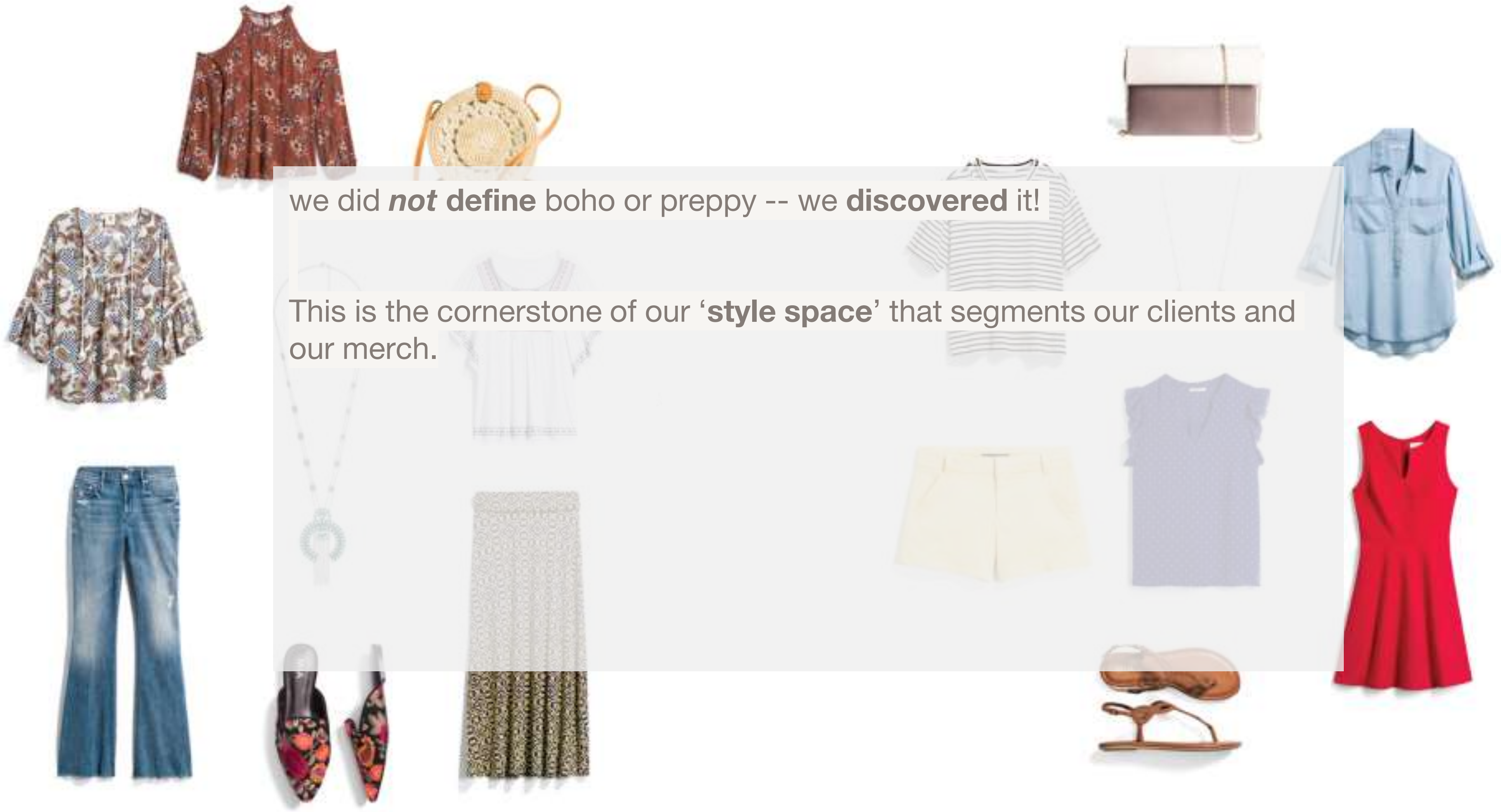
↔

striped,
polka dotted,
fitted,
button-up,
high chroma

this must be: **preppy.**

we did ***not* define** boho or preppy -- we **discovered** it!

This is the cornerstone of our ‘**style space**’ that segments our clients and our merch.



We don’t have just one dimension however….

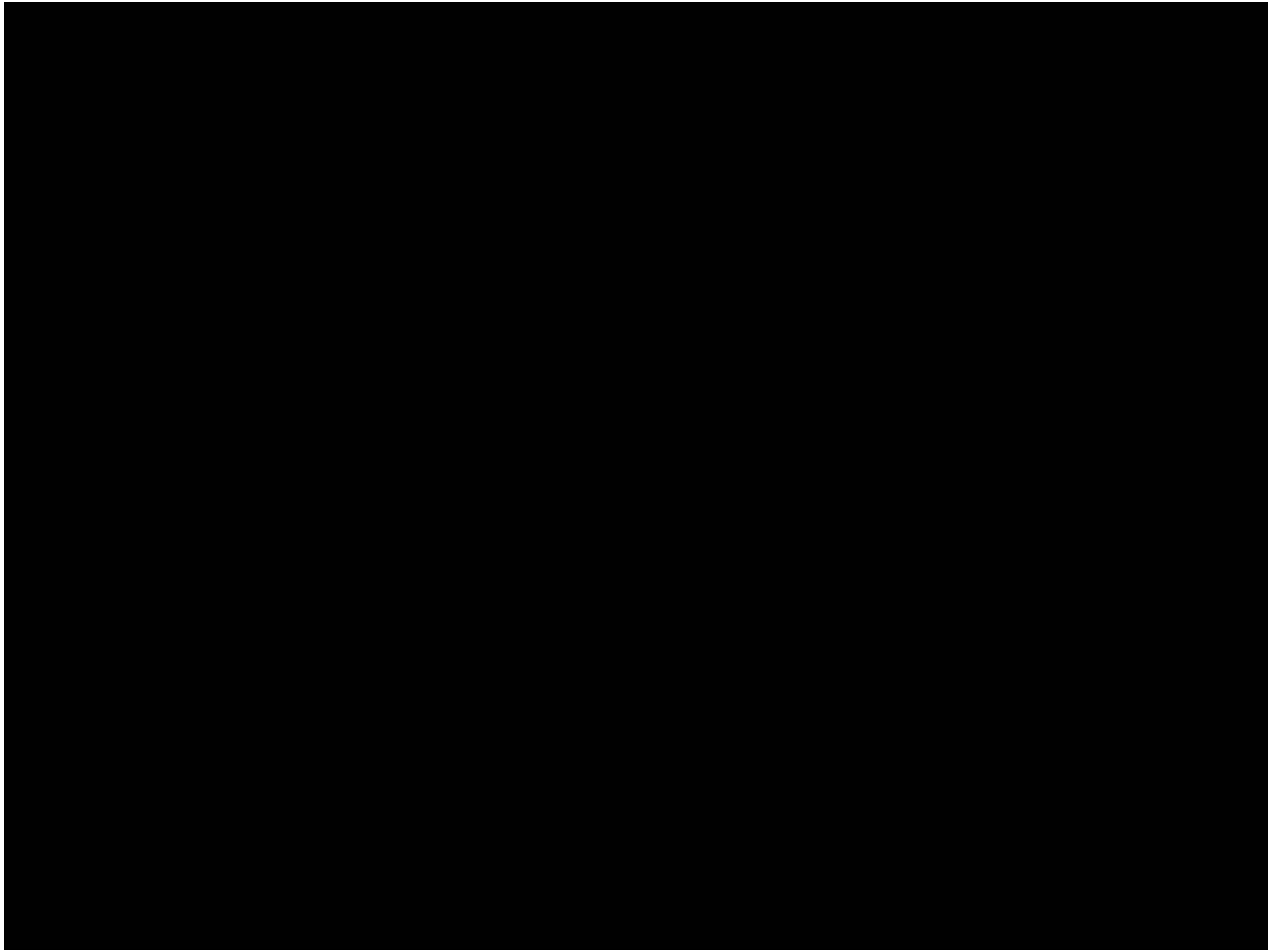
+1.2	- 0.3	+0.4	-4.0	+0.9
+6.7	- 0.44	+ 0.9	+0.58	-0.7
+3.8	- 0.9	-2. 4	+0.8	+0.3

… we have many more.

What do each of those look like?

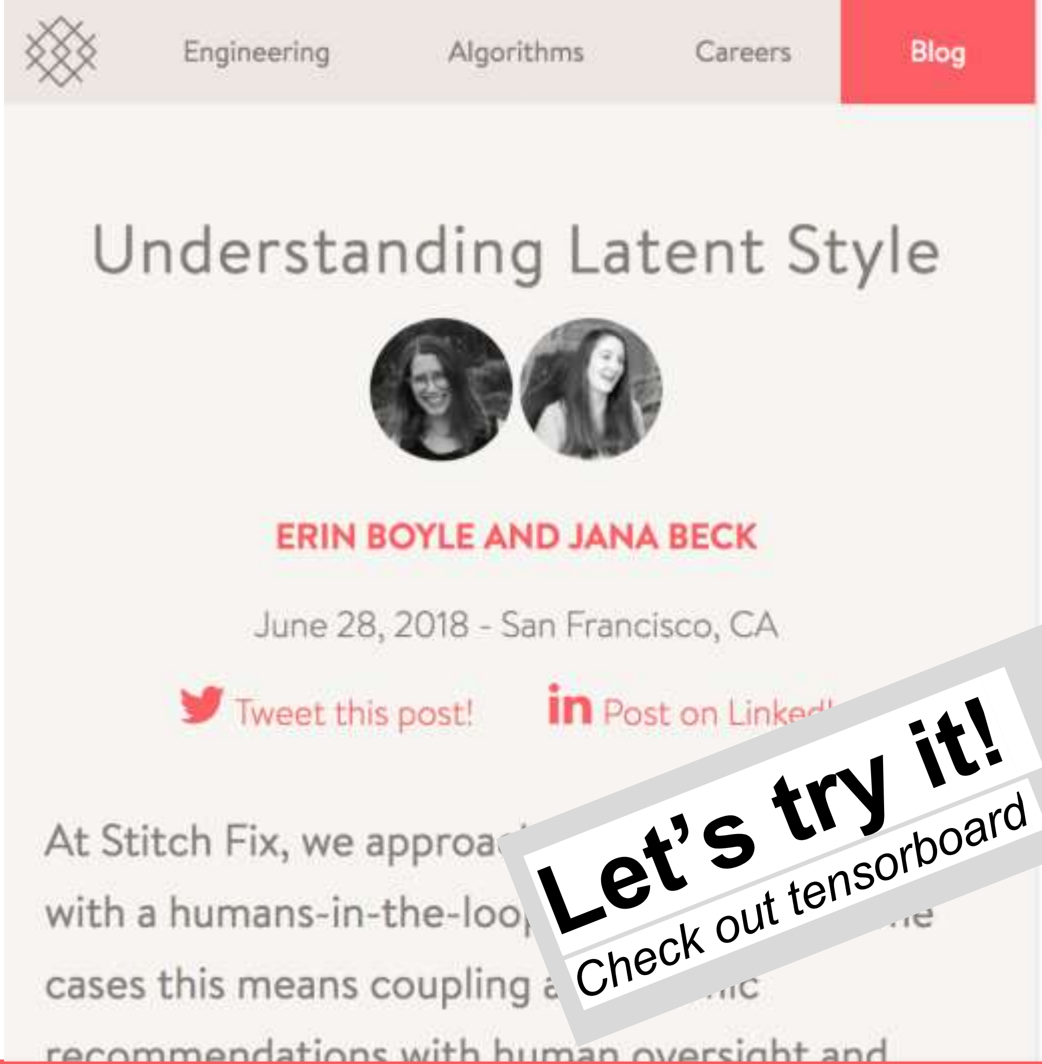
Internally, we use 18 dimensions.

The first three we call ***trendiness***, ***femininity***, and ***end use.***



For more search:
“[Understanding Latent Style](https://multithreaded.stitchfix.com/blog/2018/06/28/latent-style/)”

https://multithreaded.stitchfix.com/blog/2018/06/28/latent-style/



Let’s try it!

Check out tensorboard

Advanced Matrix Factorization

Notebooks we'll be using:

01 Training a simple MF model.ipynb
02 Simple MF Model with Biases.ipynb

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

This is now a typical matrix-factorization recommender.

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

This is now a typical matrix-factorization recommender.

What if we know more “side” features -- like the user’s occupation?

Great for when we want to “coldstart” a new user.

Note that t_o interacts with q_i

$$R_{ui} = b + d_o + \omega_u + \mu_i + (p_u + t_o) \cdot q_i$$

We have two choices for side features:

- add them as a bias
  - ⟶ Choose this if you think occupation changes like rate, but not which movies
  - ⟶ *e.g. "artists like movies more than other occupations"*
- add them as a user vector
  - ⟶ Choose this if you think occupation changes depending on the item
  - ⟶ *e.g. "realtors love real estate shows"*

*Note that t_o interacts with q_i

$$R_{ui} = b + d_o + \omega_u + \mu_i + (p_u + t_o) \cdot q_i$$

We have two choices for side features:

- add them as a bias

  → Choose this if you think occupation changes like rate, but not whic
  movies

  → *e.g. "realtors love real estate shows"*

- add them as a user vector

  → Choose this if you think occupation

  → *e.g. "realtors love real estate shows"*

Let's try it!

03 MF model with side-features

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

What about when features change in time?

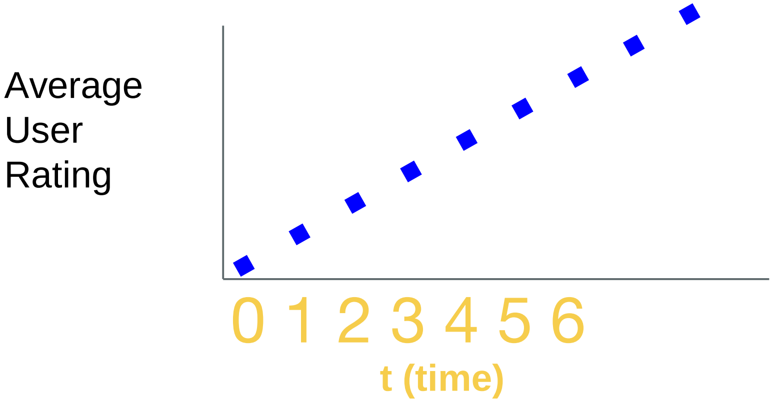
You’ve seen biases.
You’ve seen user-item interactions.

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i + m_u \cdot n_t$$

What does a latent temporal vector encode?

Here's a toy example.

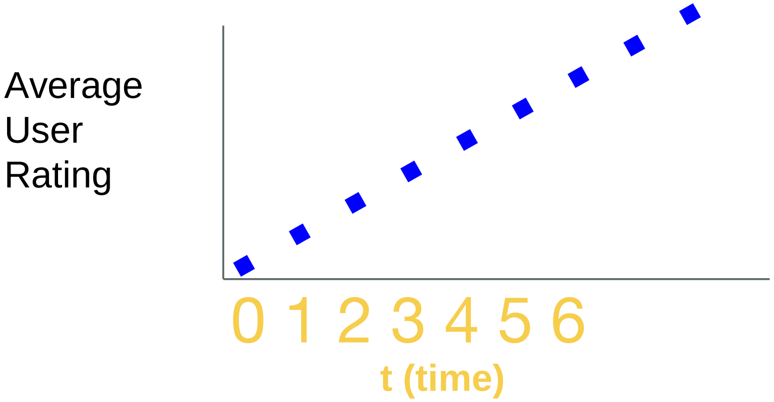
$m_0 =$  $\begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$



$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i + m_u \cdot n_t$$

What does a latent temporal vector encode?

Here's a toy example.



$m_0 =$

0.0	1.0

$n_0 =$

1.0	0.0

$n_1 =$

0.9	0.1

⋮

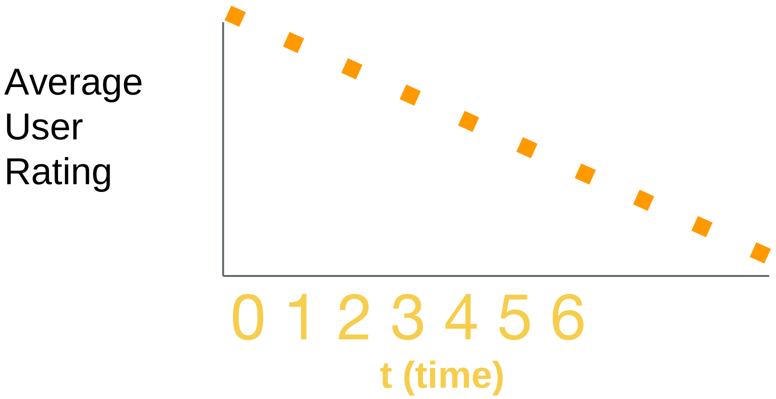
$n_t =$

1 - t	t

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i + m_u \cdot n_t$$

What does a latent temporal vector encode?

Here's a toy example.



$m_1 =$ | 1.0 | 0.0 |

$n_0 =$ | 1.0 | 0.0 |

$n_1 =$ | 0.9 | 0.1 |

⋮

$n_t =$ | 1 - t | t |

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i + m_u \cdot n_t$$

What does a latent temporal vector encode?

A user mixture of time-series.

$m_0 = \begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$

User 0 will have a 100% mixture of the 1st time series

$m_1 = \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}$

User 1 will have a 100% mixture of the 2nd time series

$n_0 = \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}$

$n_1 = \begin{bmatrix} 0.9 & 0.1 \end{bmatrix}$

$\vdots$

$n_t = \begin{bmatrix} 1 - t & t \end{bmatrix}$

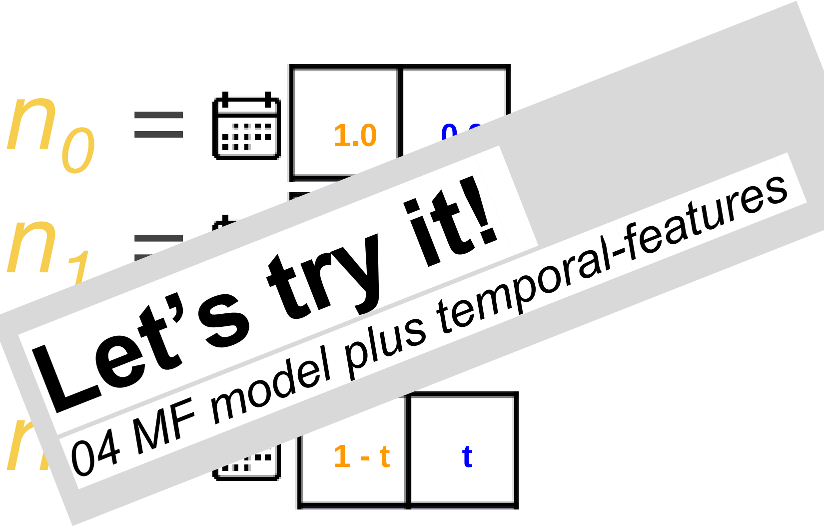
$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i + m_u \cdot n_t$$

We can enforce that nearby times have similar components.

Enforce that:

$$|n_t - n_{t-1}|$$

should be small.



Word2Vec is actually a rec engine!

Notebooks we'll be using:

03 Simple MF Biases is actually word2vec.ipynb

“**ITEM_92** I think this fabric is wonderful (rayon & spandex). like the lace/embroidery accents”

Co-occurrence modeling

w

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c

X[c, w] += 1

Co-occurrence modeling

w

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c

$X[c,\ w]$ += 1

Co-occurrence modeling

w

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c

$X[c, w] \mathrel{+}= 1$

Co-occurrence modeling

w

“ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents”

c

$$X[c,\ w] \mathrel{+}= 1$$

Co-occurrence modeling

w

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c

X[c, w] += 1

Co-occurrence modeling

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c $\qquad$ w

$$X[c,\ w] \mathrel{+}= 1$$

Co-occurrence modeling

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c w

$X[c, w] \mathrel{+}= 1$

Co-occurrence modeling

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

c w

$X[c,\ w] \mathrel{+}= 1$

Co-occurrence modeling

"ITEM_92 think fabric wonderful rayon spandex like lace embroidery accents"

c w

X[c, w] += 1

Co-occurrence modeling

w w w w w

“ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents”

c w w w w

$X[c, w]$ = count

Co-occurrence modeling

w w w w w

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

w c w w w

X[c, w] = count

Co-occurrence modeling

w w w w w

"ITEM_92 think fabric wonderful rayon

spandex like lace embroidery accents"

w w c w w

$X[c, w]$ = count

Co-occurrence modeling

w w w w w

"ITEM_92 think fabric wonderful rayon

spandex like lace **embroidery** accents"

w w w c w

$X[c, w]$ = count

Co-occurrence modeling

w w w w w

"ITEM_92 think fabric wonderful rayon spandex like lace embroidery **accents**"

w w w w c

$X[c, w]$ = count

$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

Instead of
(users, items) we have **(token1, token2)**

And instead of a
Rating we have a **skipgram count**.

But it's still the same model!

$$R_{ui} = \omega_u + \mu_i + p_u \cdot q_i$$

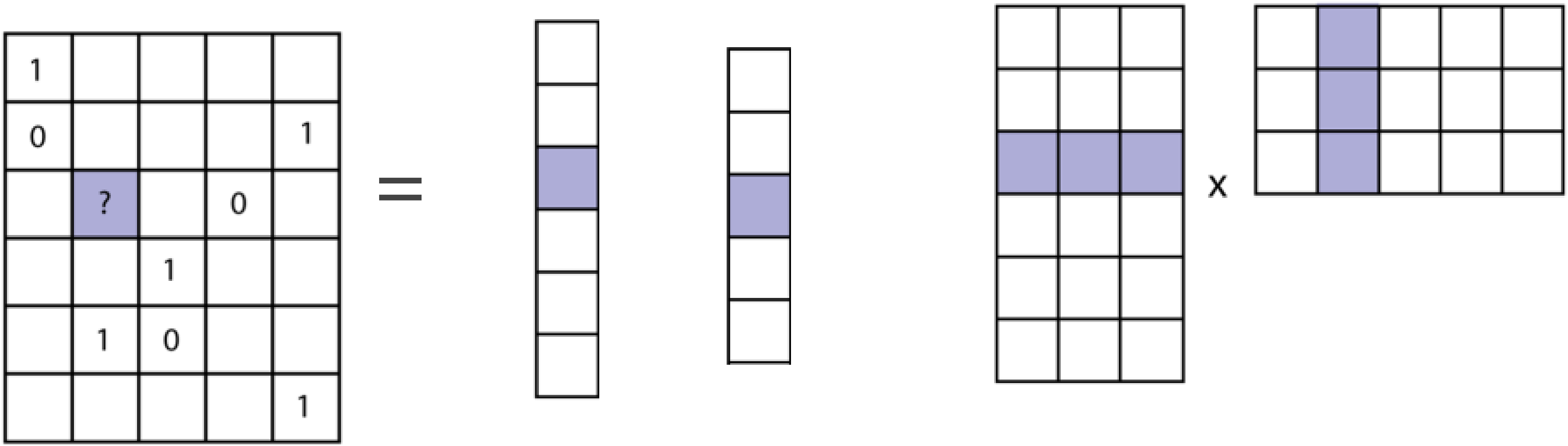
~~A single +1 or 0 rating~~
Log Skipgram count

~~User bias~~
token1 bias

~~Item bias~~
token2 bias

~~User vector~~
token1 vector

~~Item vector~~
token2 vector



$$R_{ui} = \omega_u + \mu_i + p_u \cdot q_i$$

~~A single +1 or 0 rating~~
Log Skipgram count

~~User bias~~
token1 bias

(How frequent is this word?)

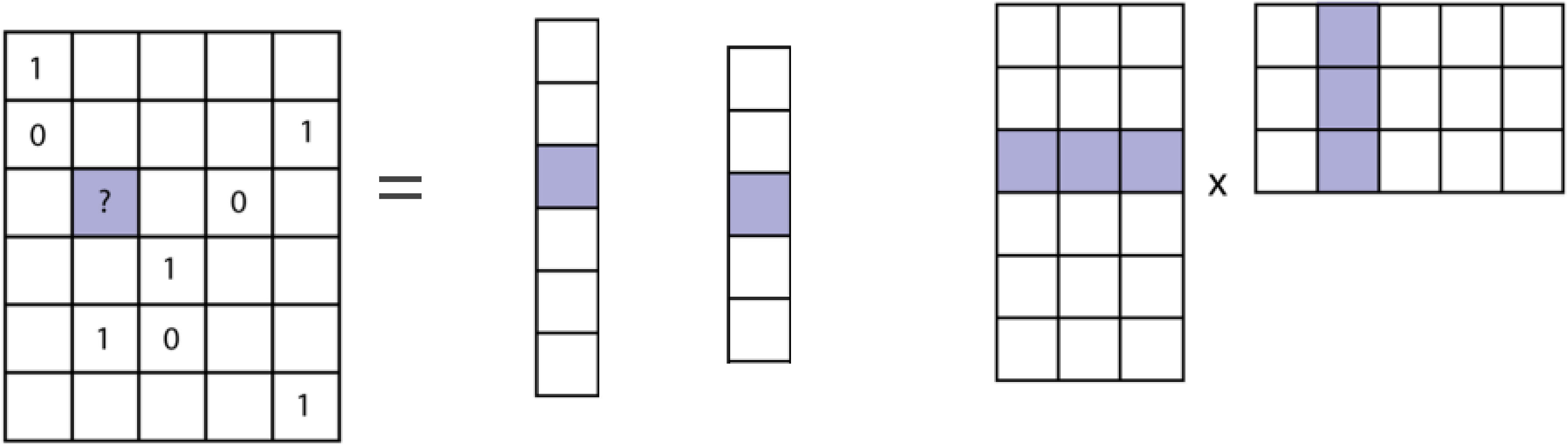
~~Item bias~~
token2 bias

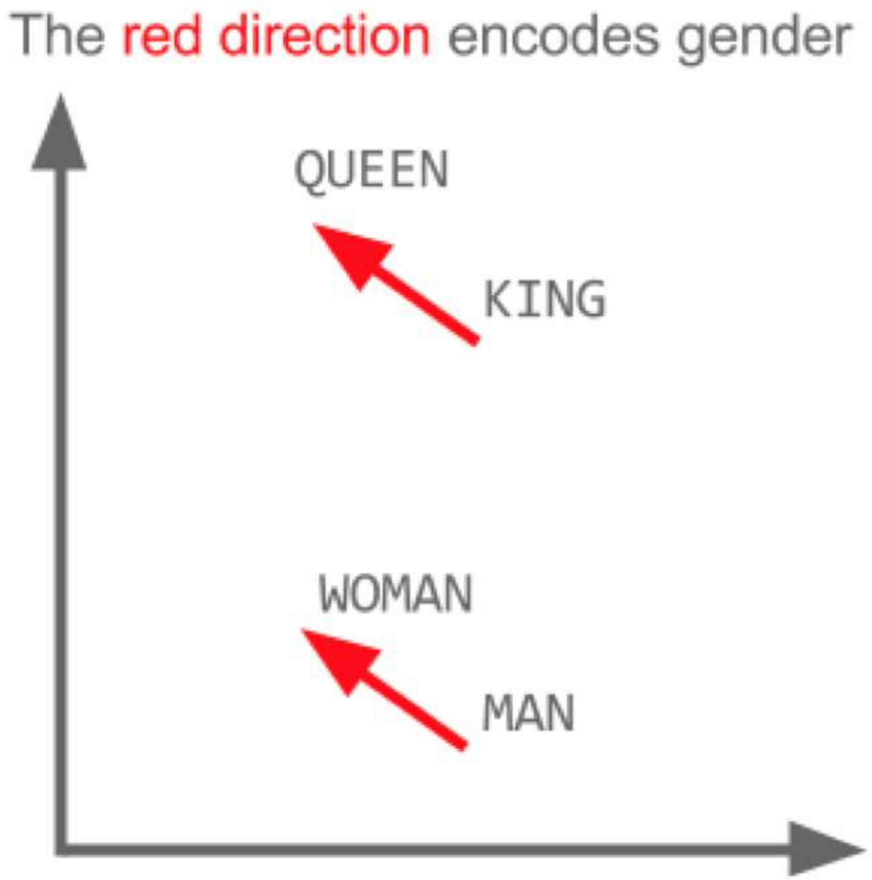
(How frequent is this word?)

~~User vector~~
word1 vector

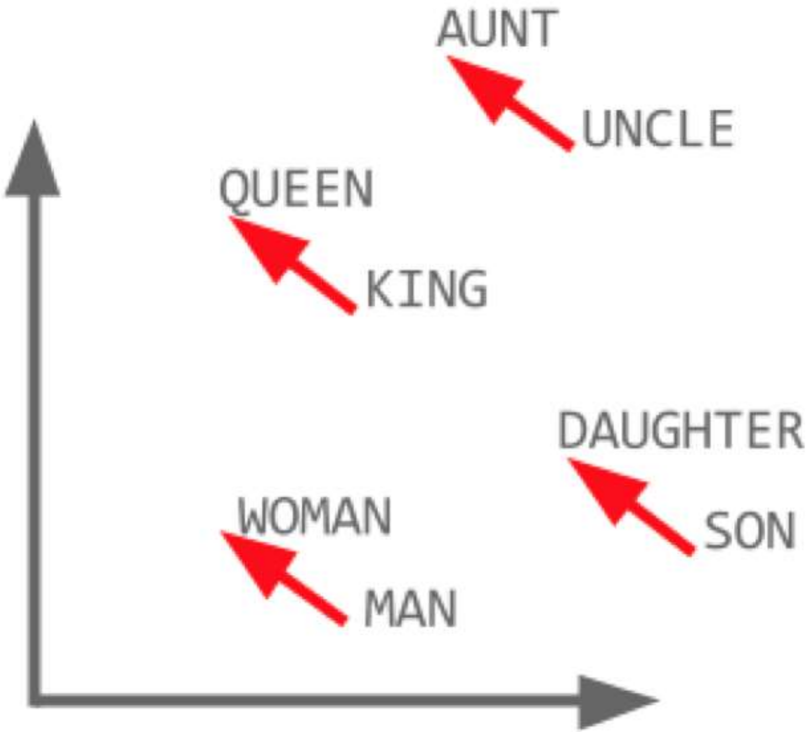
~~Item vector~~
word2 vector

Do word1 & word2 have a special interaction?

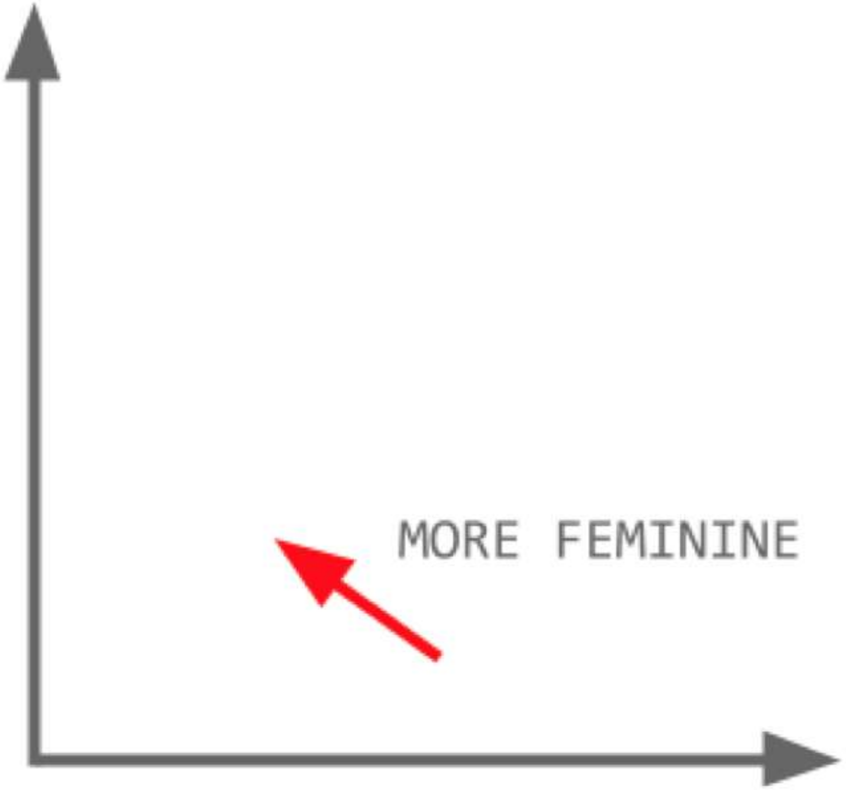



The red direction encodes gender
QUEEN
KING
WOMAN
MAN

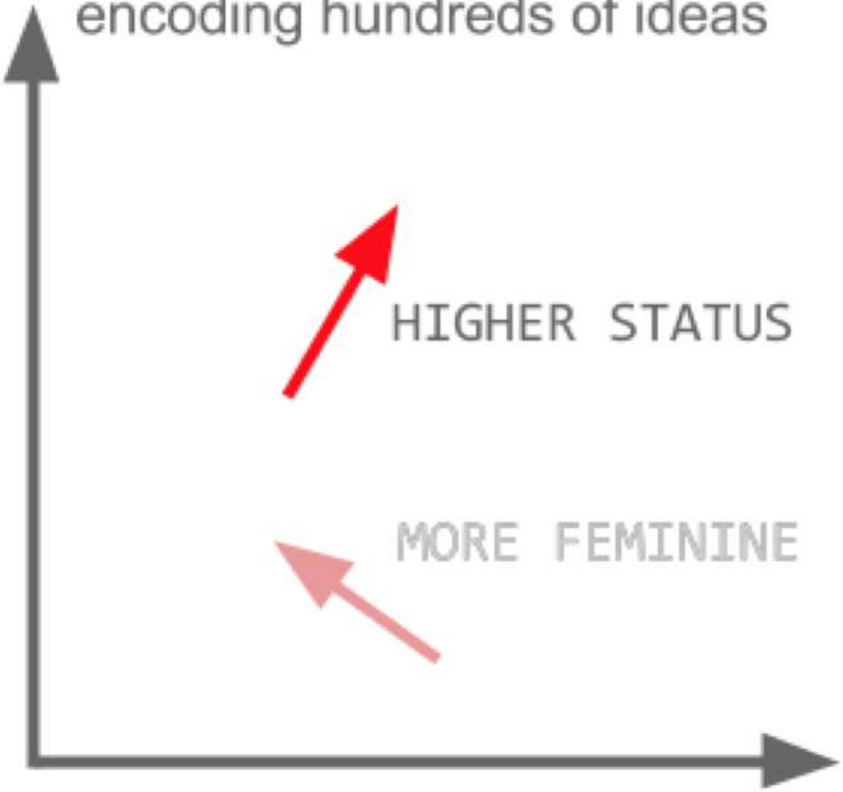
Which is consistent across all words



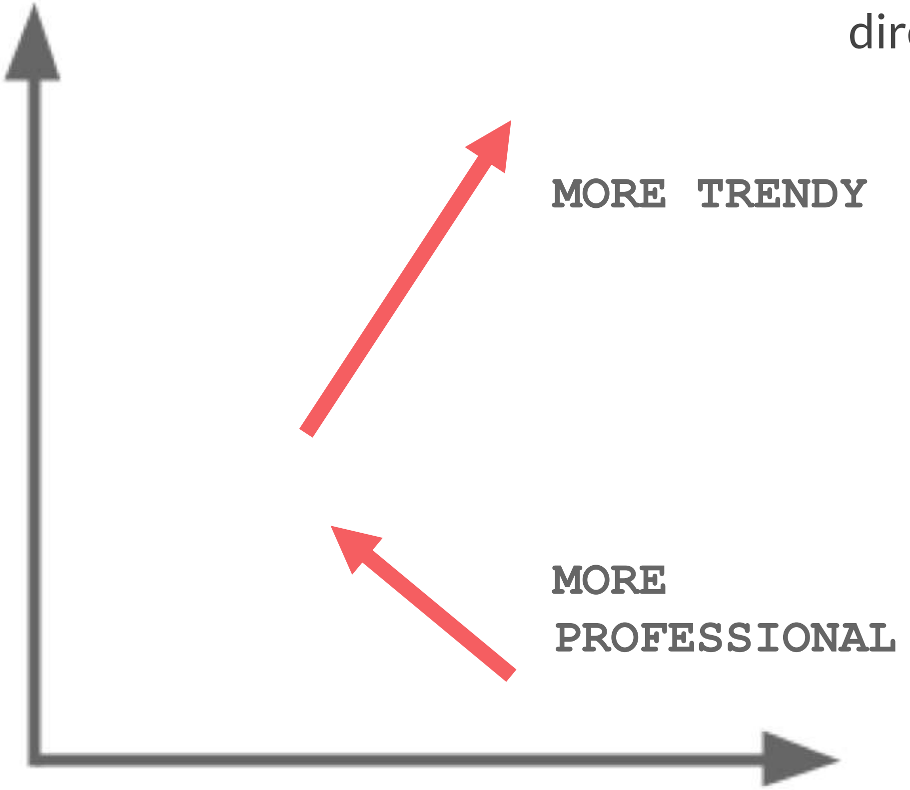
This direction always means gender



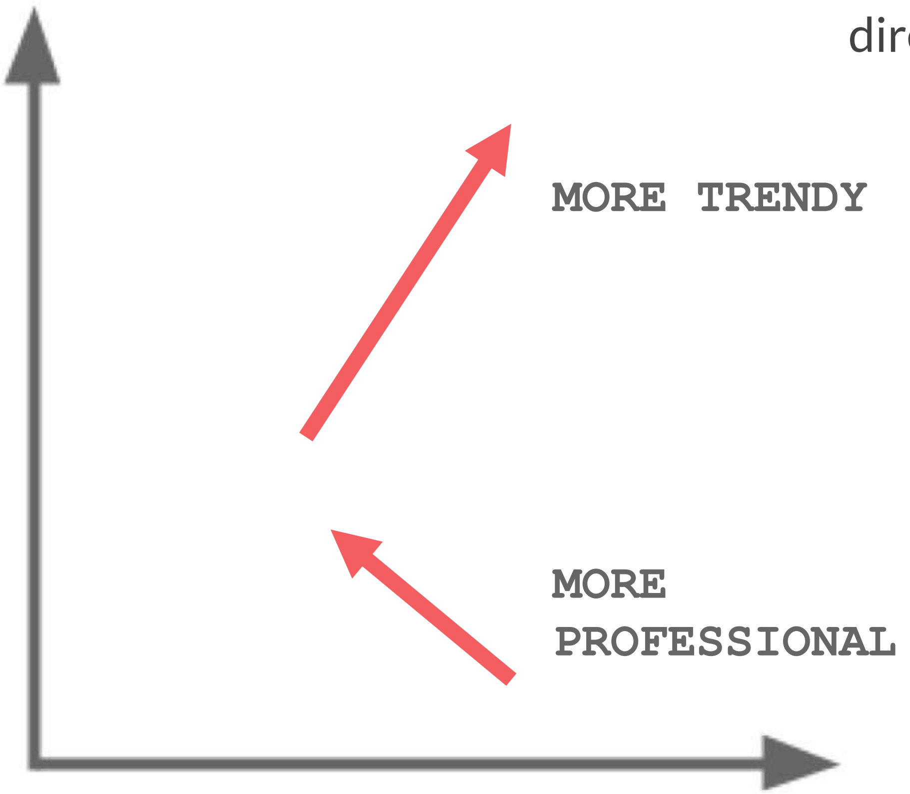
We have hundreds of directions encoding hundreds of ideas

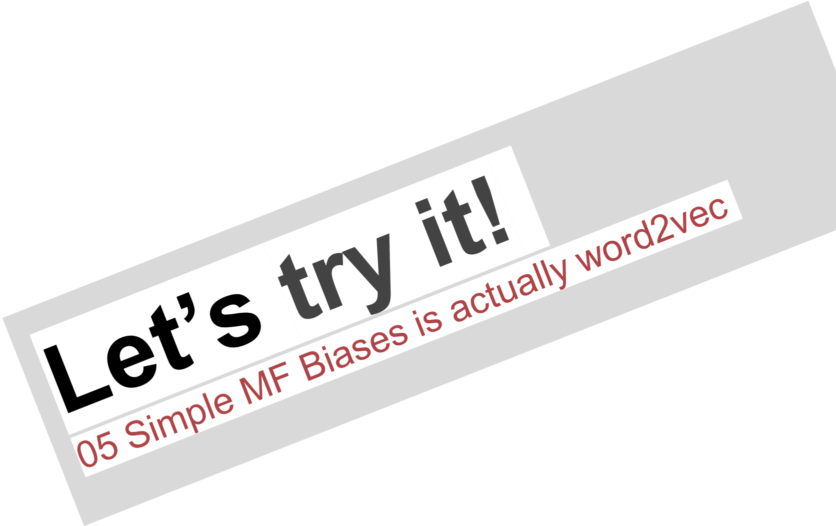


In recommendation engines, there's also directions in the latent space. What are the directions in your space?



In recommendation engines, there's also directions in the latent space. What are the directions in your space?





Variational Matrix Factorization

Notebooks we'll be using:

08 Variational MF.ipynb

Practical reasons to go variational:

1. Alternative regularization
2. Measure what your model ***doesn't know***.
3. Help explain your data.

Practical reasons to go variational:

1. Alternative regularization
2. Measure what your model ***doesn't know***.
3. Help explain your data.
4. **Short & fits in a tweet**!

Ryan Adams
@ryan_p_adams

Following

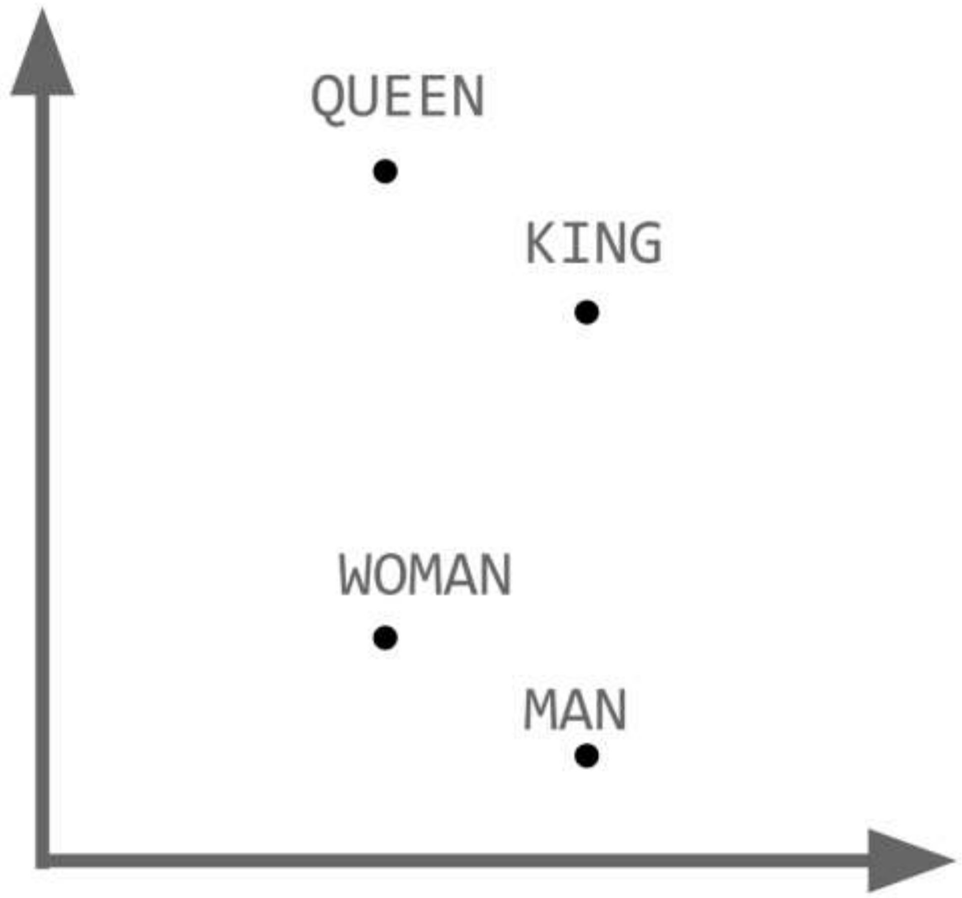
@DavidDuvenaud

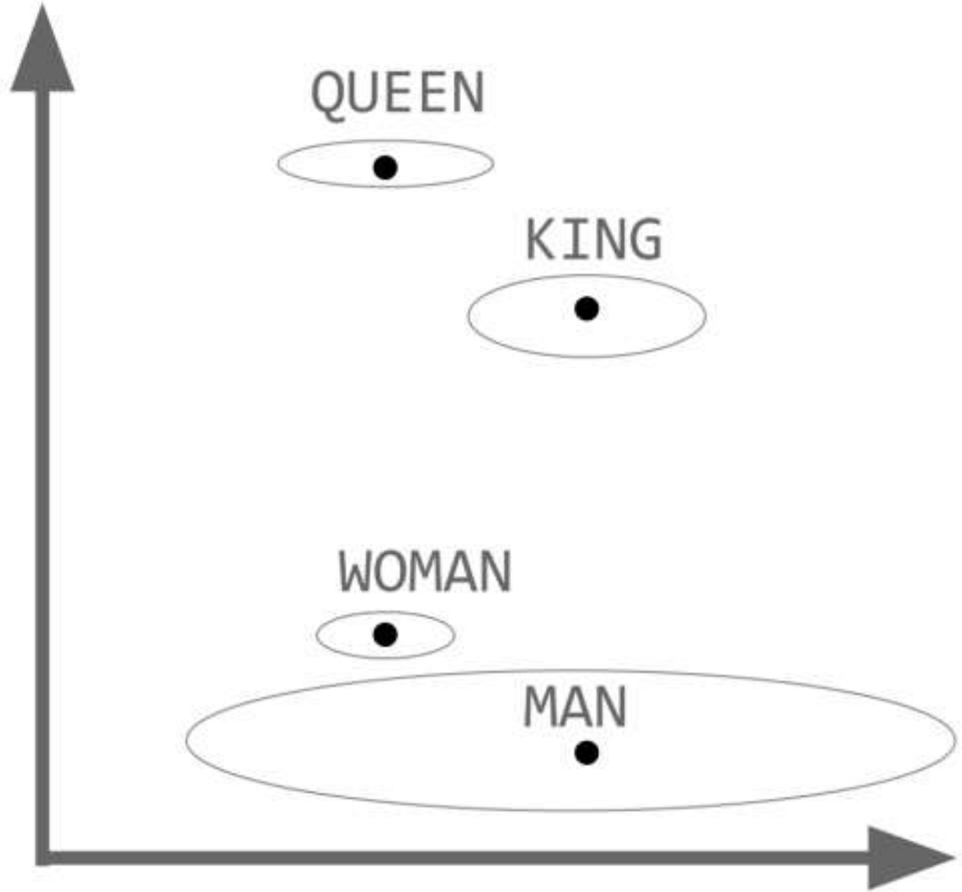
```
def elbo(p, lp, D, N):
 v=exp(p[D:])
 s=randn(N,D)*sqrt(v)+p[:D]
 return mvn.entropy(0, diag(v))+mean(lp(s))
gf = grad(elbo)
```

RETWEETS 7 LIKES 22

9:43 AM - 7 Nov 2015

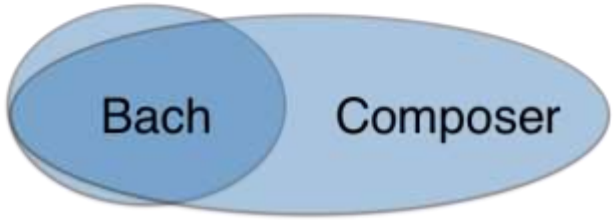
1 7 22


QUEEN
KING
WOMAN
MAN

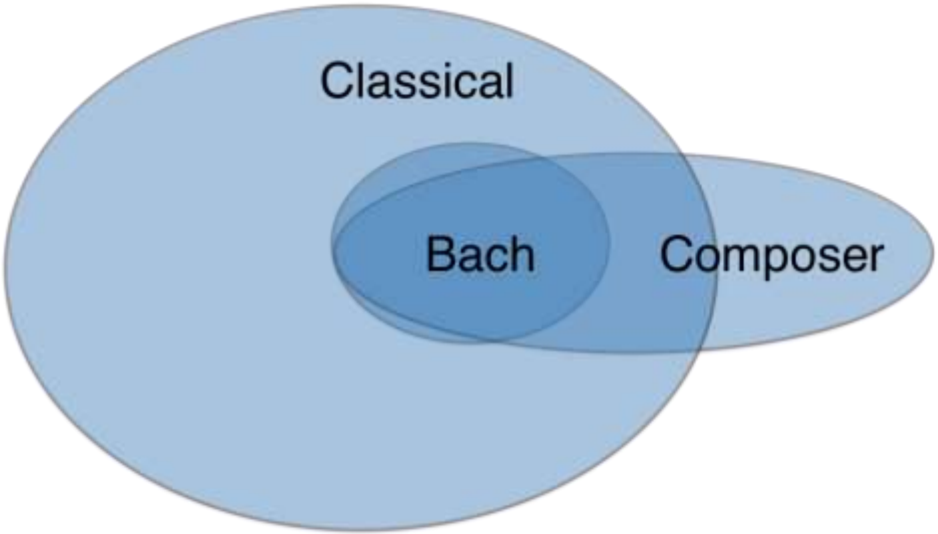

QUEEN
KING
WOMAN
MAN

See also:
‘word2gauss’

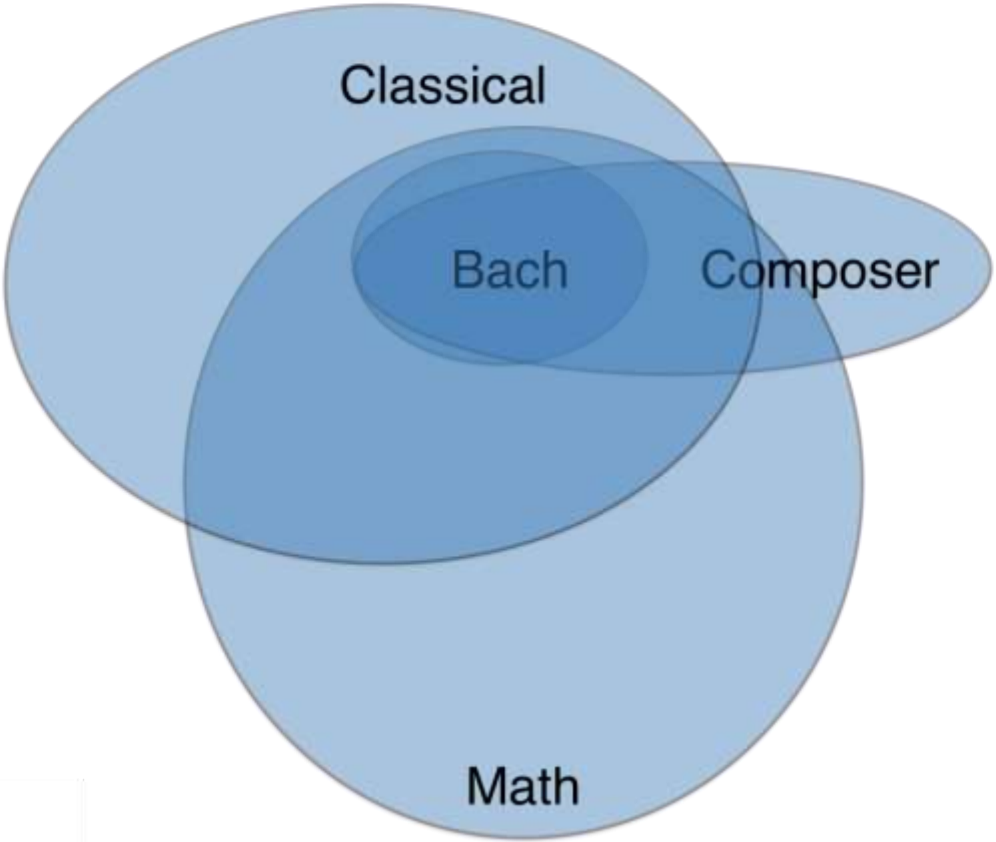
See also:
'word2gauss'



See also:
‘word2gauss’



See also:
‘word2gauss’



$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

Let’s make this variational:

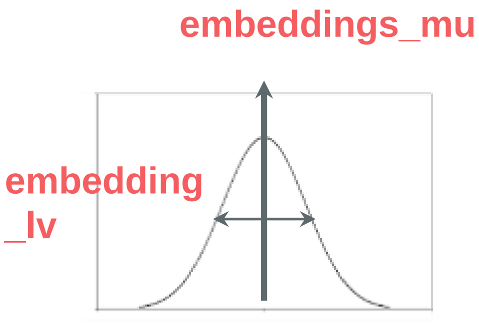
1. **Replace point estimates with samples from a distribution.**
2. Replace regularizing that point, regularize that distribution.

WITHOUT VARIATIONAL

```
embeddings = nn.Embedding(n_users, k)

c_vector = embeddings(c_index)
```

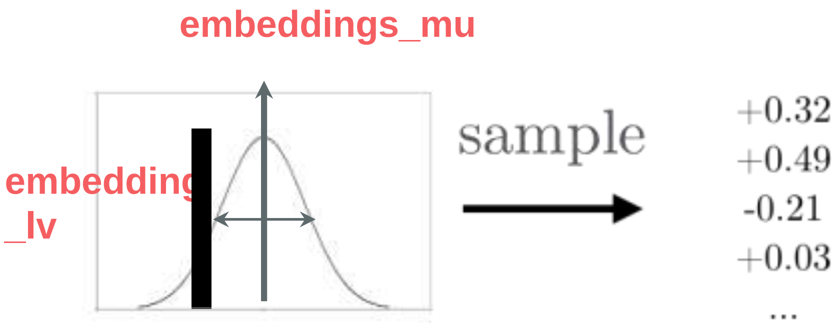
WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)


c_vector = sample_gaussian(vector_mu, vector_lv)
```



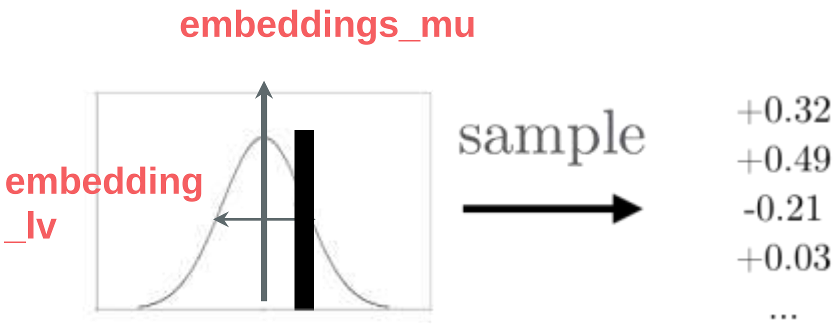
WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)


c_vector = sample_gaussian(vector_mu, vector_lv)
```



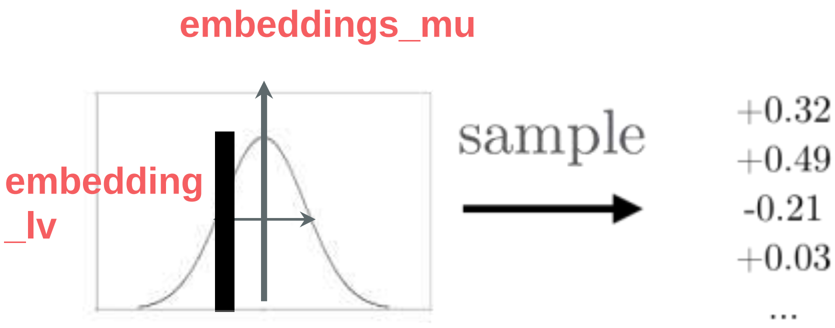
WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)



c_vector = sample_gaussian(vector_mu, vector_lv)
```



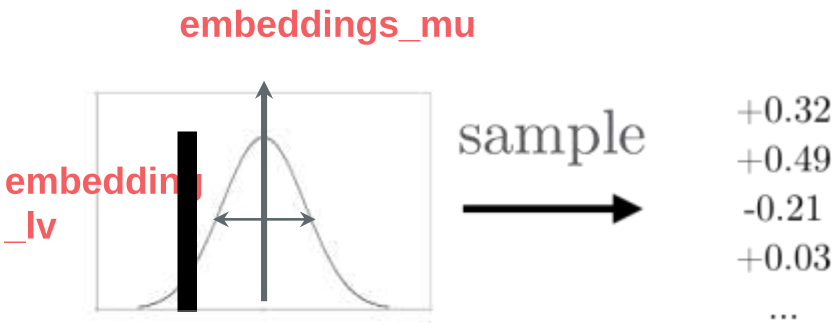
WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)



c_vector = sample_gaussian(vector_mu, vector_lv)
```



WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)



c_vector = sample_gaussian(vector_mu, vector_lv)
```



WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)


c_vector = sample_gaussian(vector_mu, vector_lv)
```

WITH VARIATIONAL

```
embeddings_mu = nn.Embedding(n_users, k)

embeddings_lv = nn.Embedding(n_users, k)

...

vector_mu = embeddings_mu(c_index)

vector_lv = embeddings_lv(c_index)



c_vector = sample_gaussian(vector_mu, vector_lv)
```

```
def sample_gaussian(mu, lv):

    variance = sqrt(exp(lv))

    sample = mu + N(0, 1) * variance

    return sample
```

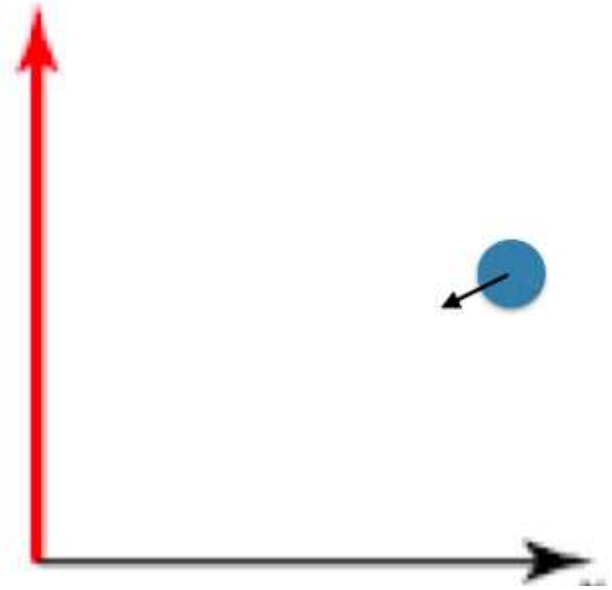
$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

Let’s make this variational:

1. Replace point estimates with samples from a distribution.
2. **Replace regularizing that point with regularizing that distribution.**

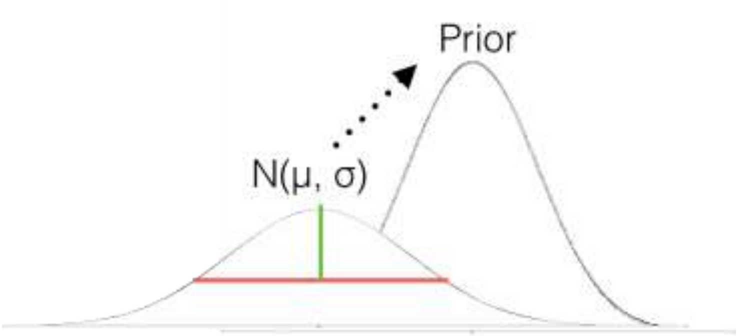
WITHOUT VARIATIONAL

loss += c_vector.pow(2.0).sum()



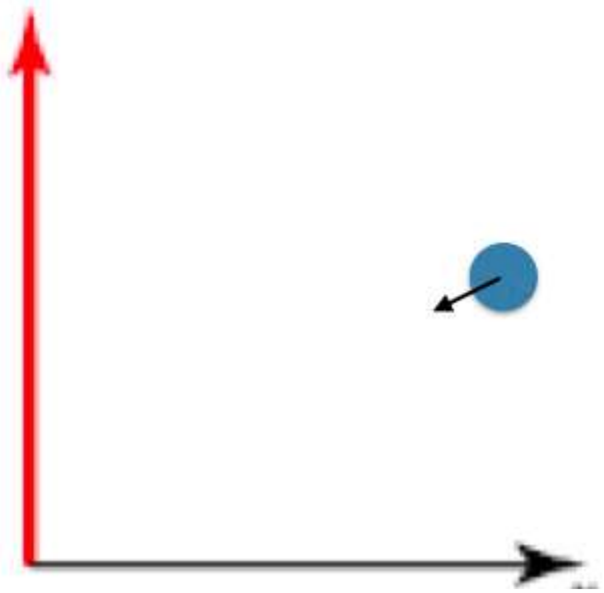
WITH VARIATIONAL

loss += gaussian_kldiv(vector_mu, vector_lv)



WITHOUT VARIATIONAL

```
loss += c_vector.pow(2.0).sum()
```



WITH VARIATIONAL

```
loss += gaussian_kldiv(vector_mu, vector_lv)
```

Let's try it!

06 Variational MF

At the Frontier

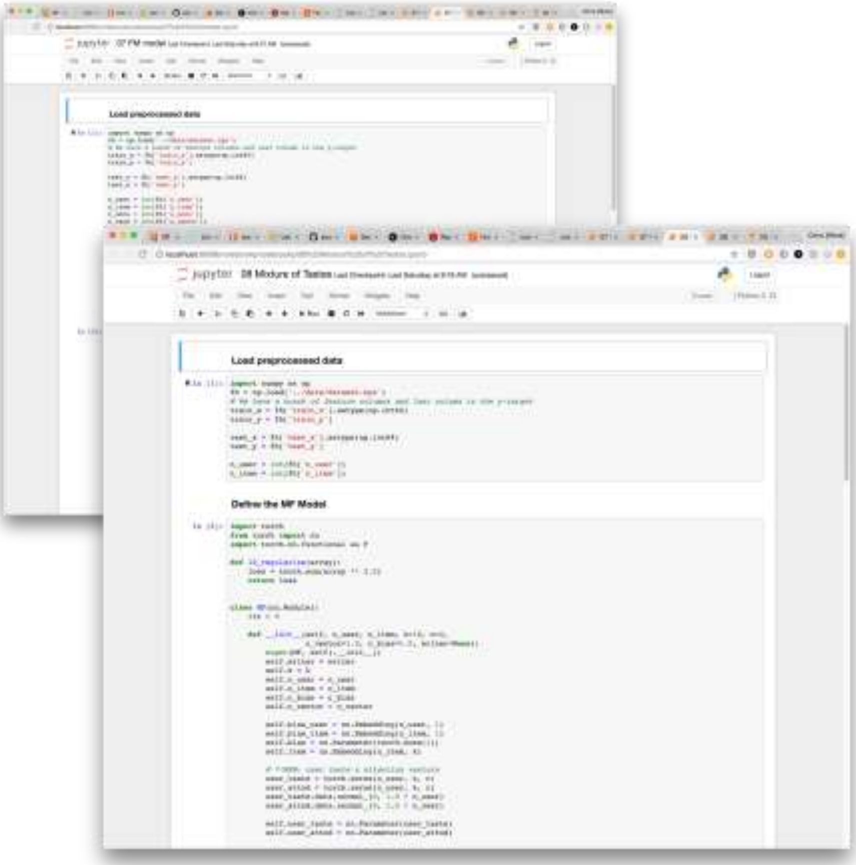
More Things to Try

FMs

- Useful when you have many *kinds* of interactions, not just user-item and user-time

Mixture-of-Tastes

- Like “attention” for recommendations, allows for users to have multiple “tastes.”



Extra: Non-Euclidean Spaces

Poincare Spaces

Cornell University Library

arXiv.org > cs > arXiv:1705.08039

Search or Article ID inside arXiv | All papers | Broade

(Help | Advanced search)

Computer Science > Artificial Intelligence

Poincaré Embeddings for Learning Hierarchical Representations

Maximilian Nickel, Douwe Kiela

(Submitted on 22 May 2017 (v1), last revised 26 May 2017 (this version, v2))

Representation learning has become an invaluable approach for learning from symbolic data such as text and graphs. However, while complex symbolic datasets often exhibit a latent hierarchical structure, state-of-the-art methods typically learn embeddings in Euclidean vector spaces, which do not account for this property. For this purpose, we introduce a new approach for learning hierarchical representations of symbolic data by embedding them into hyperbolic space -- or more precisely into an n-dimensional Poincar\'e ball. Due to the underlying hyperbolic geometry, this allows us to learn parsimonious representations of symbolic data by simultaneously capturing hierarchy and similarity. We introduce an efficient algorithm to learn the embeddings based on Riemannian optimization and show experimentally that Poincar\'e embeddings outperform Euclidean embeddings significantly on data with latent hierarchies, both in terms of representation capacity and in terms of generalization ability.

Probably the coolest idea I've seen this year.

Volume = 1

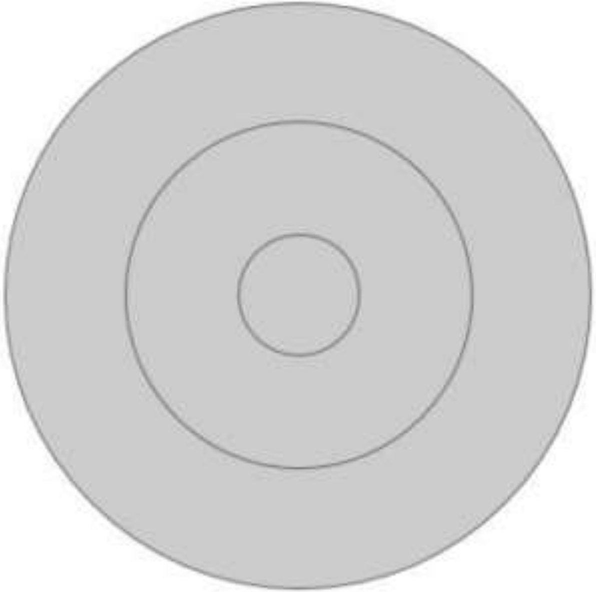
In Euclidean space, volume grows like:

$$V \sim r^d$$

Volume = 4

In Euclidean space, volume grows like:

$$V \sim r^d$$



Volume = 9

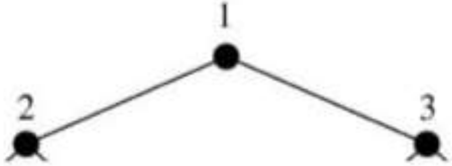
In Euclidean space, volume grows like:

$$V \sim r^d$$

Volume = 1

In binary data structures 'volume' grows like:

$$V \sim 2^r$$

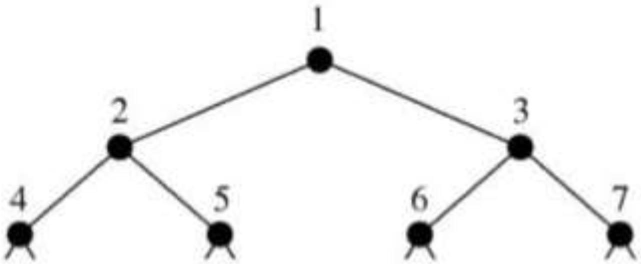


Volume = 2 + 1

In binary data structures ‘volume’ grows like:

$$V \sim 2^r$$

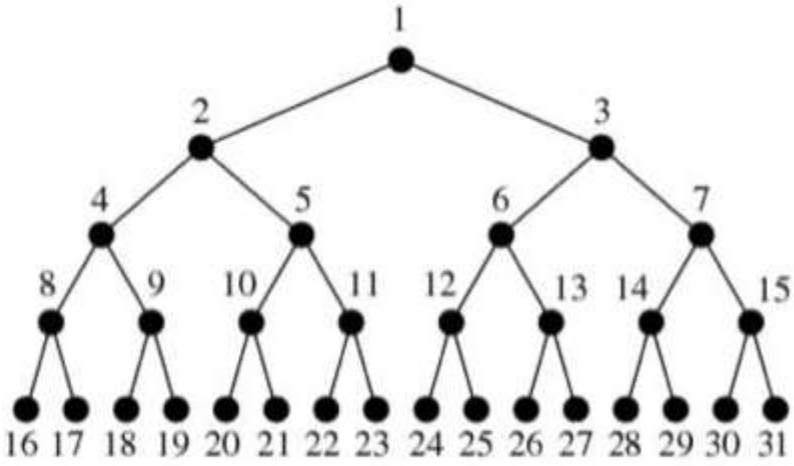
Volume = 4 + 3



In binary data structures ‘volume’ grows like:

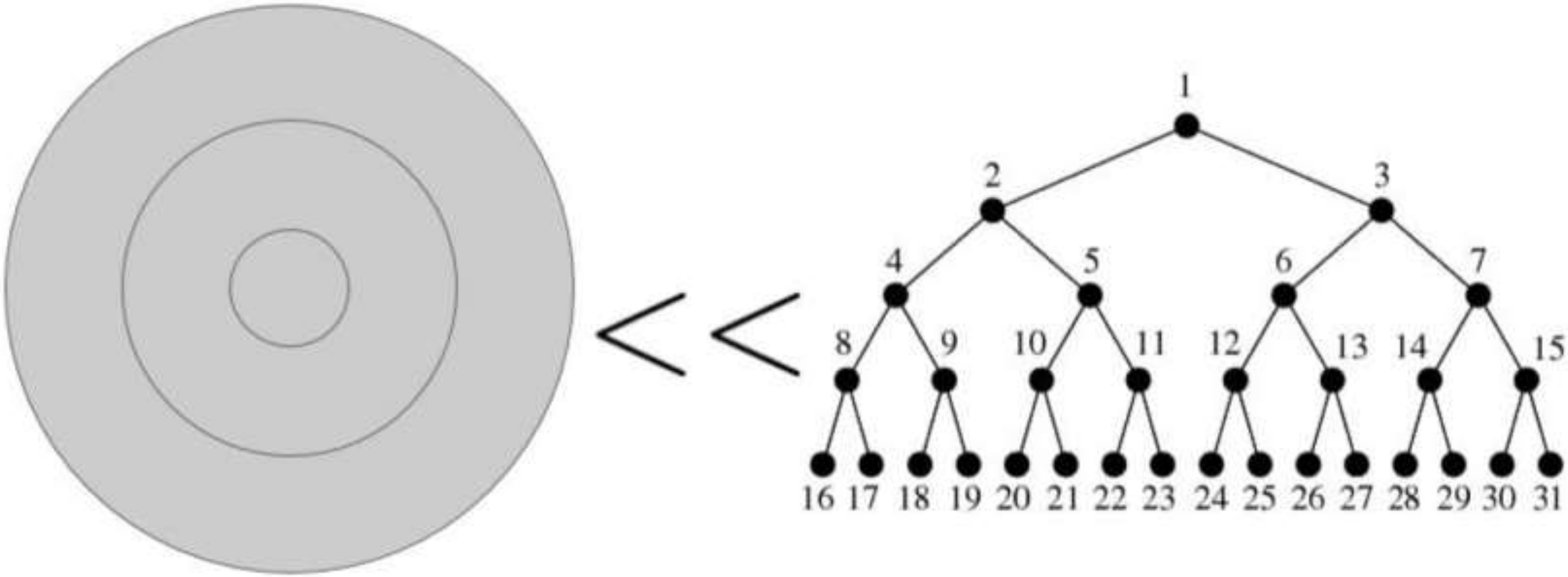
$$V \sim 2^r$$

Volume = 8 + 7



In binary data structures 'volume' grows like:

$$V \sim 2^r$$



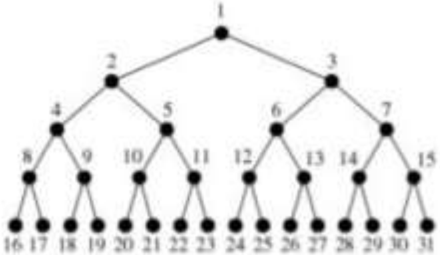
In Euclidean space, volume grows like:

$$V \sim r^d$$

In binary data structures 'volume' grows like:

$$V \sim 2^r$$

If you want to encode a hierarchy....



$$V \sim 2^r$$

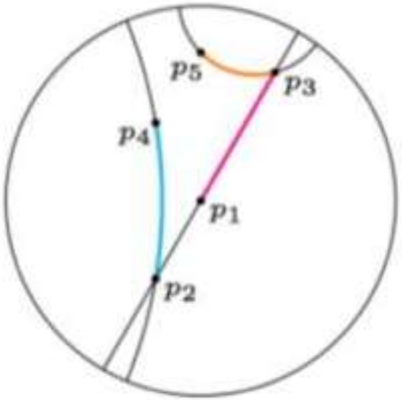
....in a euclidean space

$$V \sim r^d$$

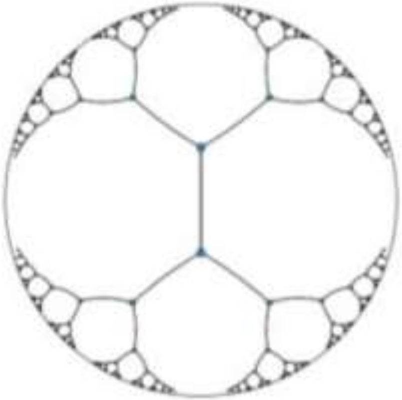
You need dimensionality that grows exponentially!

In **hyperbolic spaces** (like Poincare space) the volume grows much faster with radius.

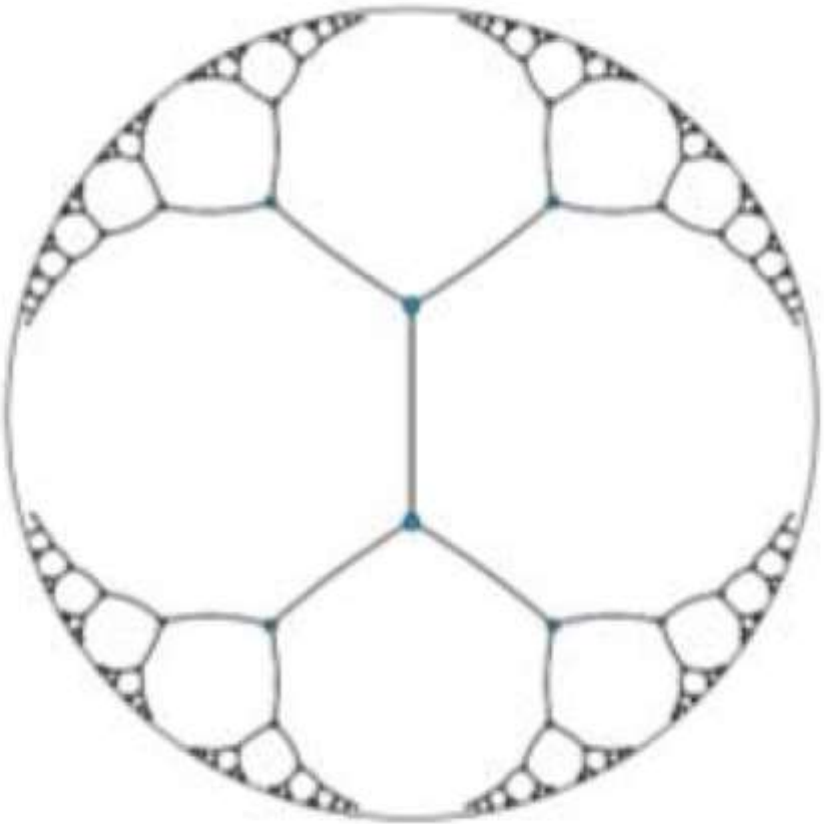
By analogy, hyperbolic spaces are continuous versions of hierarchies.



(a) Geodesics of the Poincaré disk



(b) Embedding of a tree in $\mathcal{B}^2$



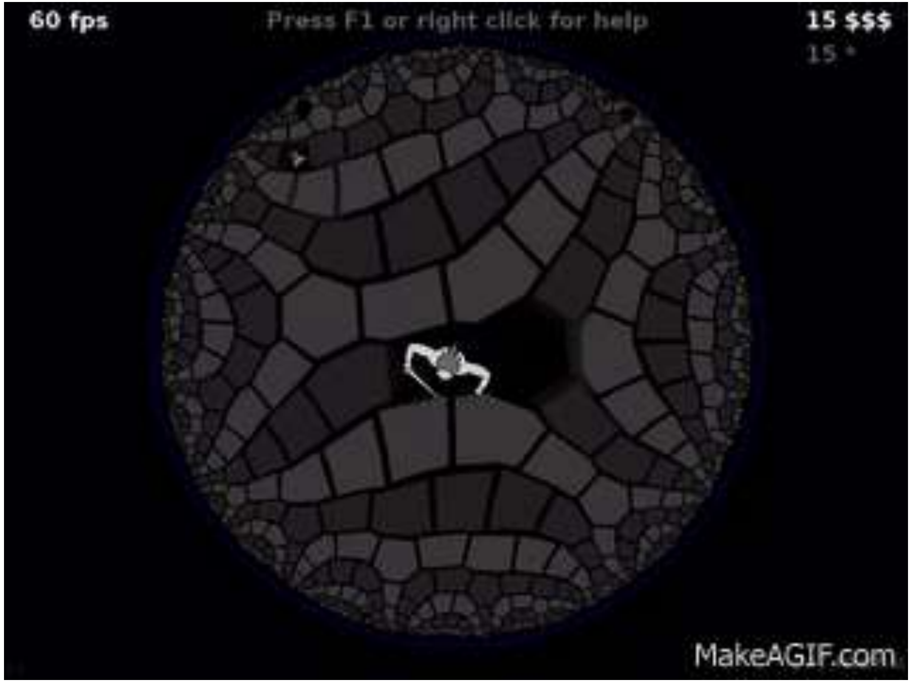
Binary Tree embedded in Poincaré space

“Zelda” is a game in set 2D Euclidean space (roughly).

- Infinity can’t be seen
- Area is homogenous
- Geodesics are straight lines

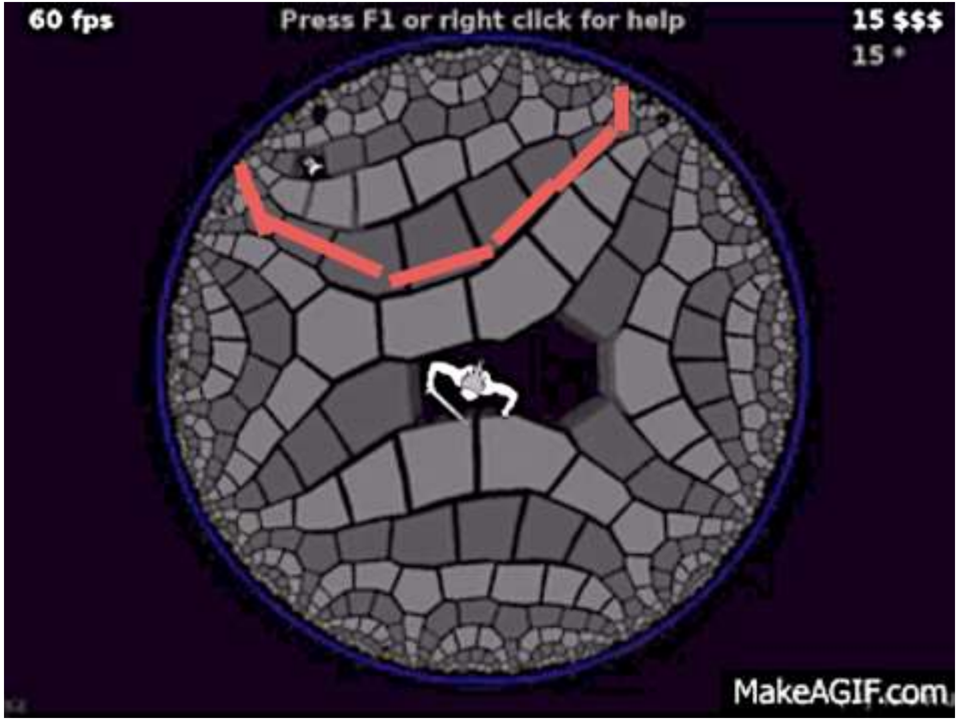
“Hyper Rogue” is a game in hyperbolic space.

- Radius = 1 is infinitely far away, but visible
- Volume increases towards boundary

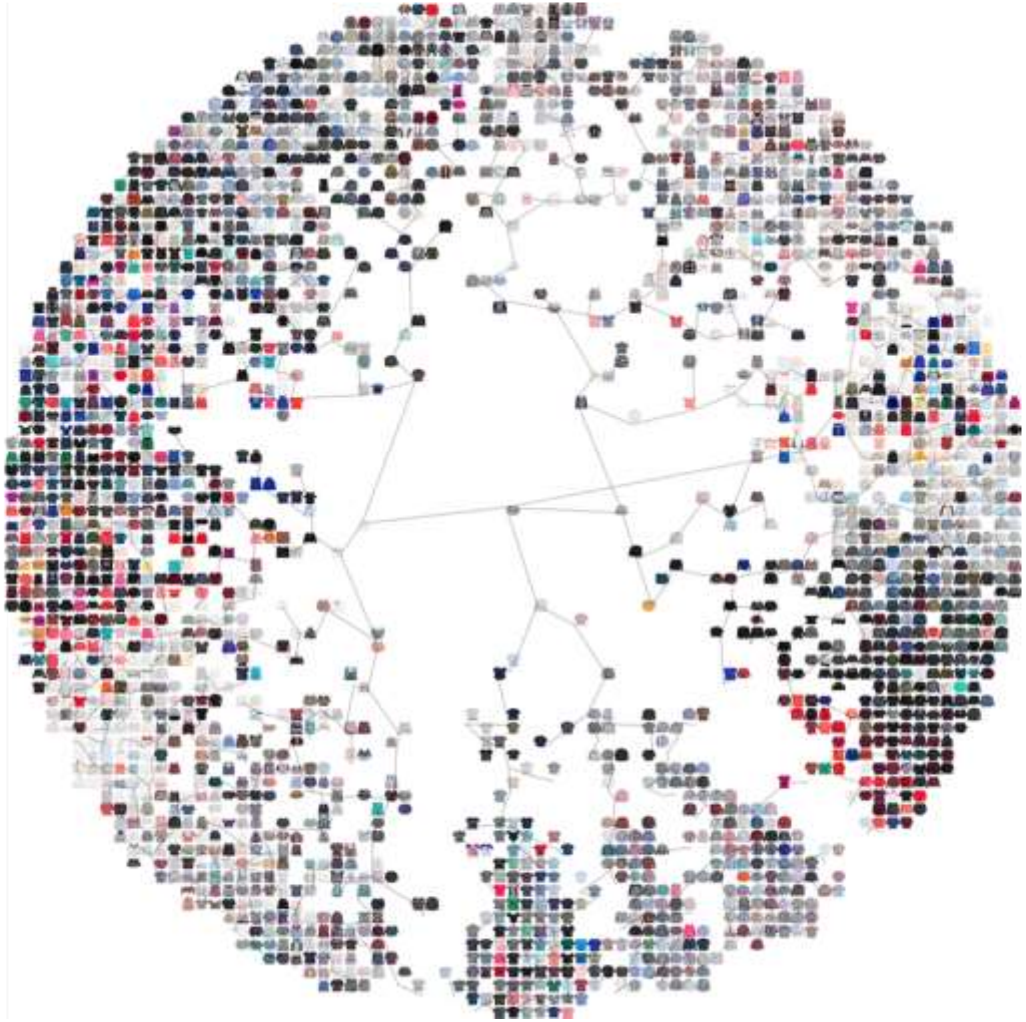


“Hyper Rogue” is a game in hyperbolic space.

- Radius = 1 is infinitely far away, but visible
- Volume increases towards boundary



Hierarchical Graph Structure of our styles in Poincaré-SNE



?

S

@chrisemoody

Stitch Fix

MultiThreaded