

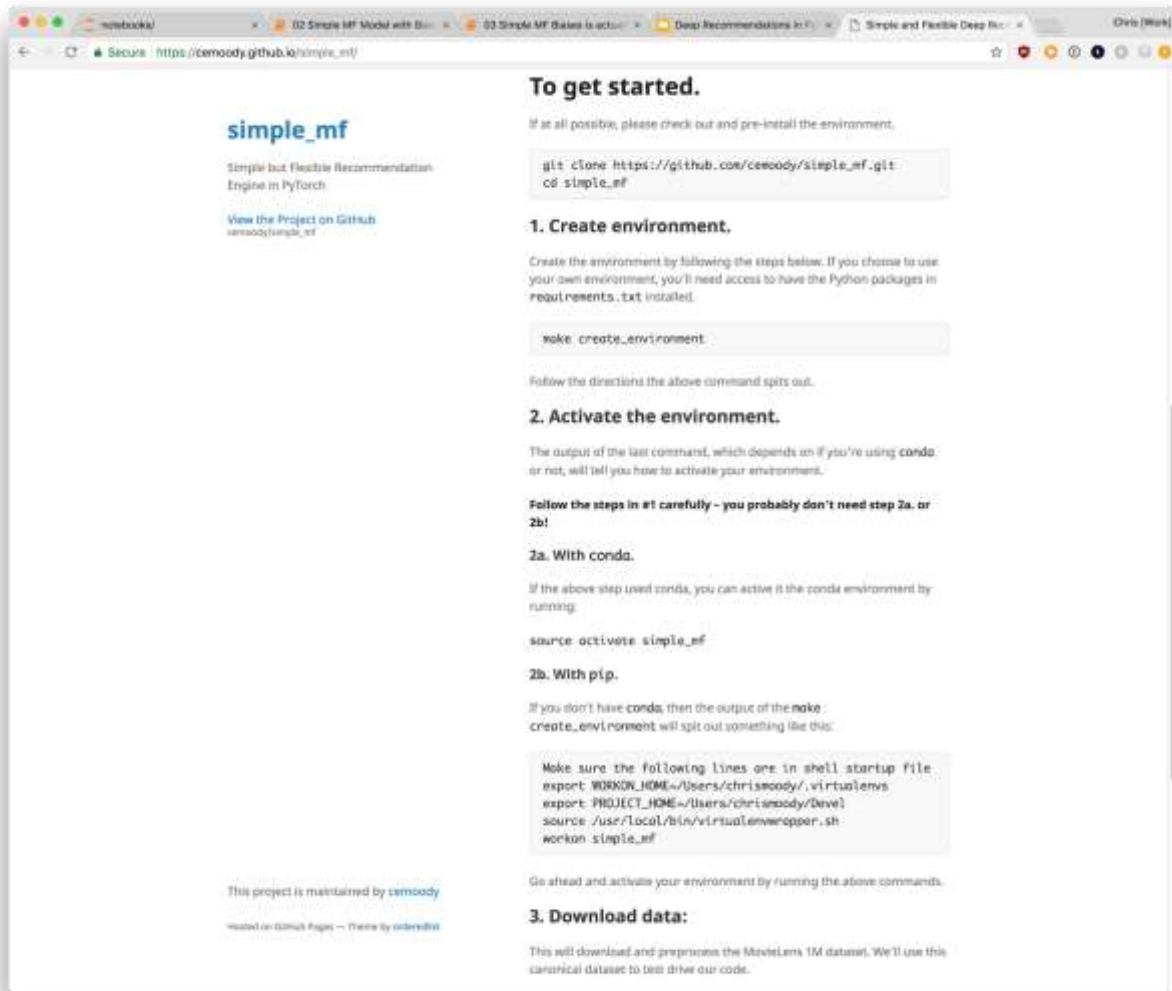


# Deep Recommendations in PyTorch

# Setup

Follow along with instructions here:

[cemoodys.github.io/simple\\_mf](https://cemoodys.github.io/simple_mf)



The screenshot shows the GitHub repository page for `simple_mf` by `cemoodys`. The page is titled "simple\_mf" and describes it as a "Simple but Flexible Recommendation Engine in PyTorch". It provides instructions on how to get started, including cloning the repository, creating and activating the environment, and downloading data.

**simple\_mf**  
Simple but Flexible Recommendation Engine in PyTorch  
[View the Project on GitHub](#)  
[cemoodys/simple\\_mf](#)

**To get started.**  
If at all possible, please check out and pre-install the environment.

```
git clone https://github.com/cemoodys/simple_mf.git
cd simple_mf
```

**1. Create environment.**  
Create the environment by following the steps below. If you choose to use your own environment, you'll need access to have the Python packages in `requirements.txt` installed.

```
make create_environment
```

Follow the directions the above command spits out.

**2. Activate the environment.**  
The output of the last command, which depends on if you're using `conda` or not, will tell you how to activate your environment.

**Follow the steps in it carefully - you probably don't need step 2a. or 2b!**

**2a. With conda.**  
If the above step used `conda`, you can active it the `conda` environment by running:

```
source activate simple_mf
```

**2b. With pip.**  
If you don't have `conda`, then the output of the `make create_environment` will spit out something like this:

```
Make sure the following lines are in shell startup file
export WORKON_HOME=~/.virtualenvs
export PROJECT_HOME=~/.virtualenvs/Devel
source /usr/local/bin/virtualenvwrapper.sh
workon simple_mf
```

Go ahead and activate your environment by running the above commands.

**3. Download data:**  
This will download and preprocess the MovieLens 1M dataset. We'll use this canonical dataset to test drive our code.

This project is maintained by [cemoodys](#)  
Hosted on GitHub Pages — Theme by [orderloli](#)

# About



@chrisemoody



Caltech Physics



PhD. in Astrophysics + Supercomputing

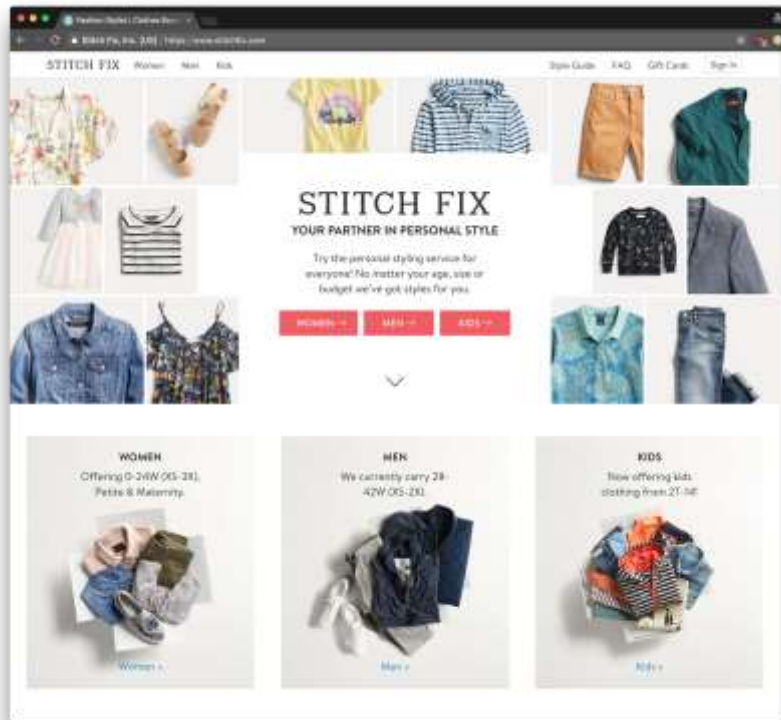


Scikit-Learn t-SNE Contributor

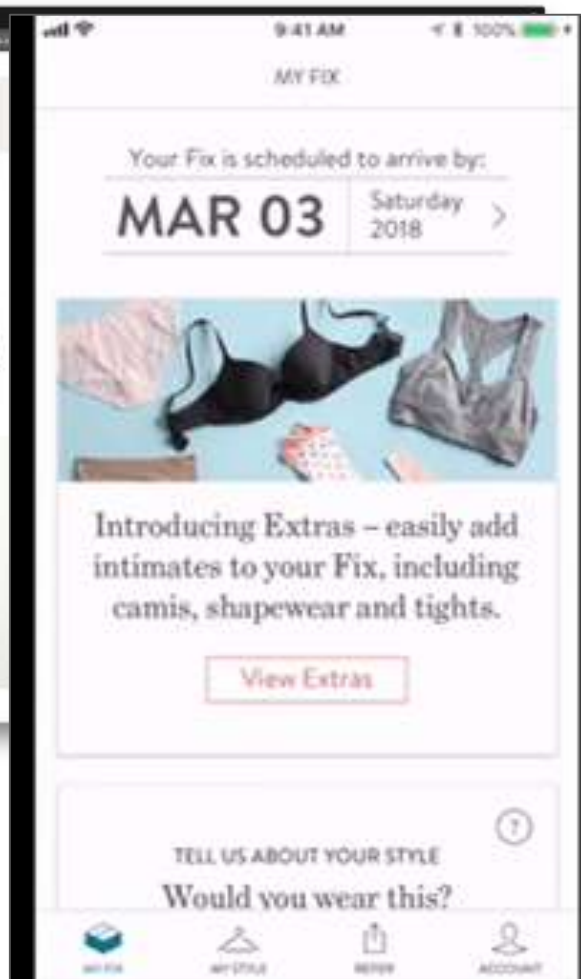


Stitch Fix AI Team

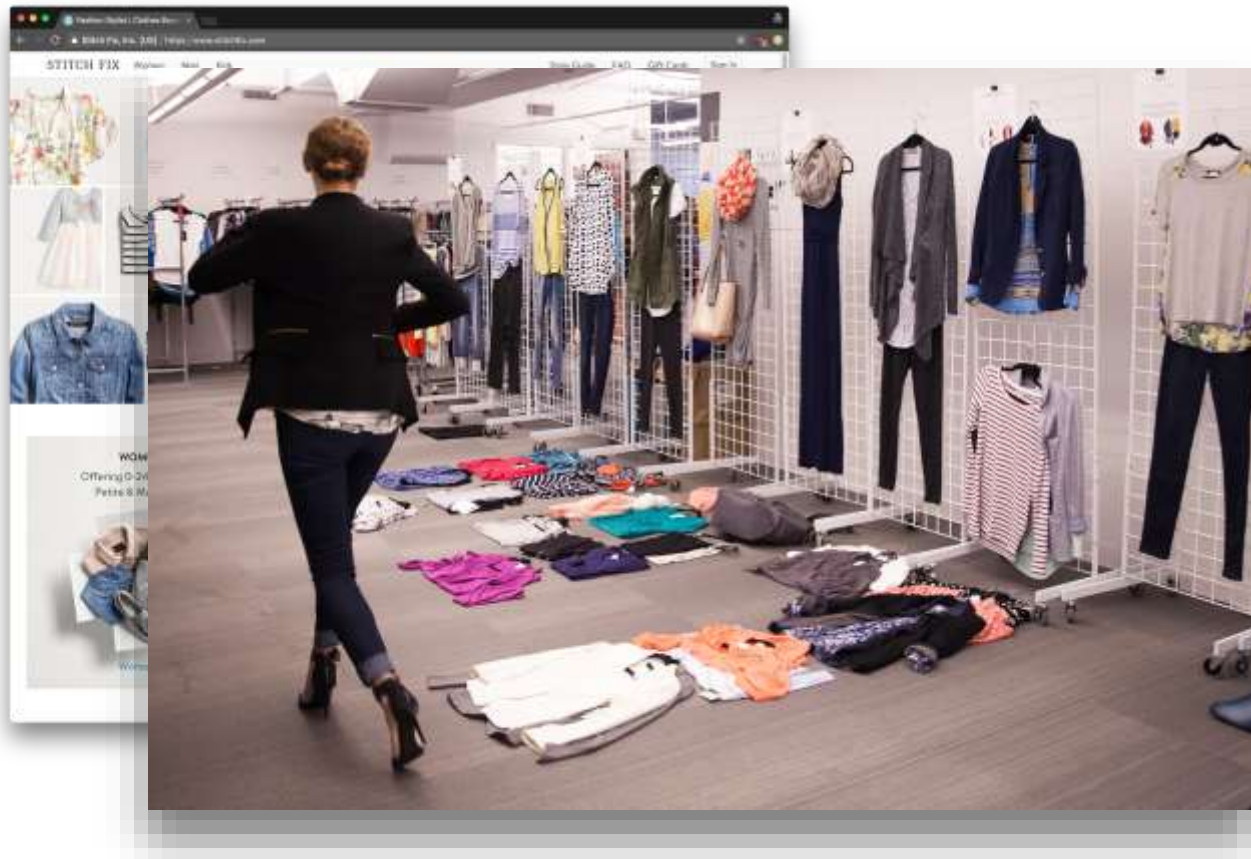
# Stitch Fix



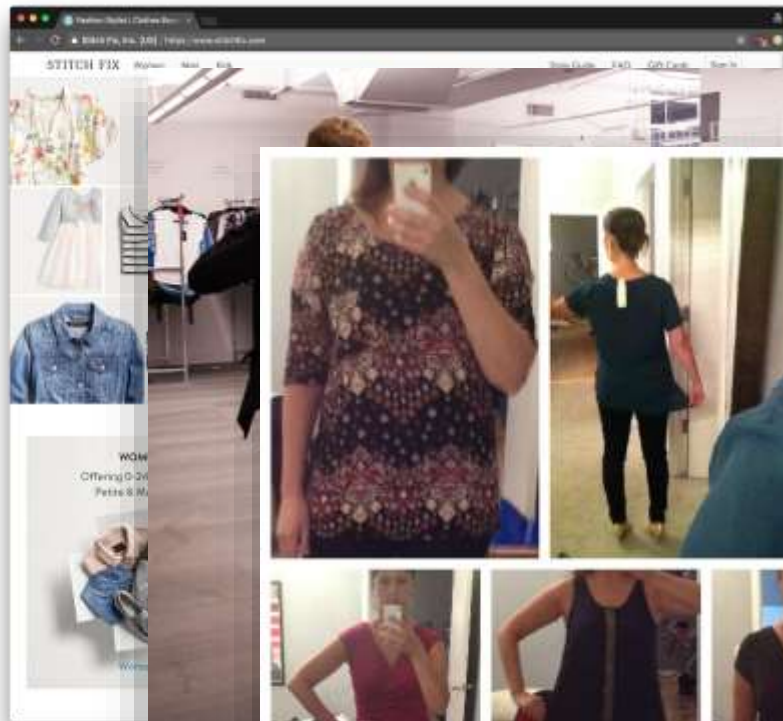
# Stitch Fix



# Stitch Fix

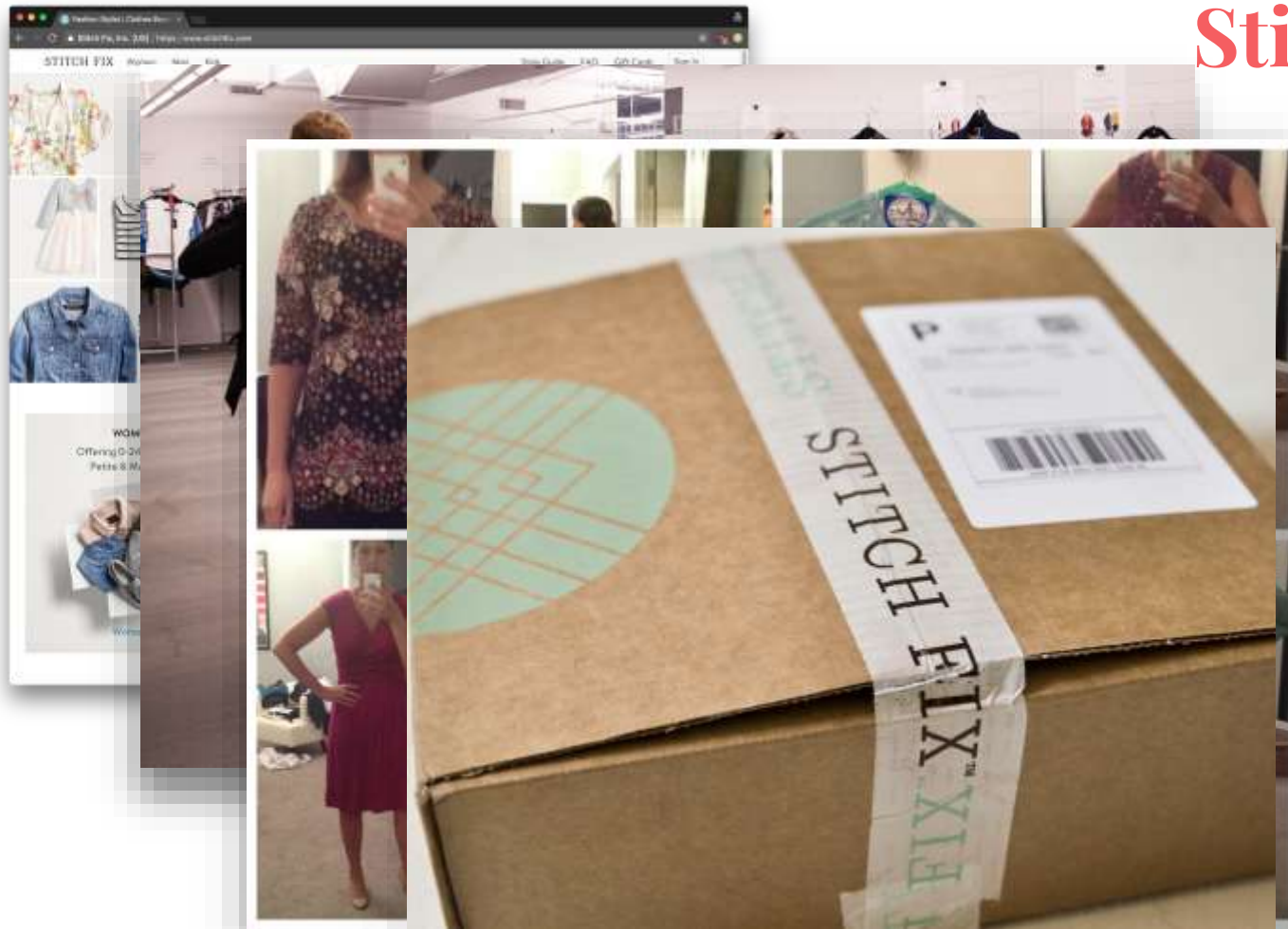


# Stitch Fix





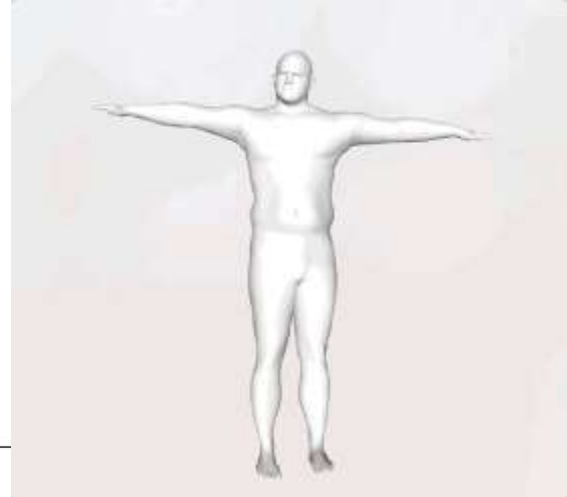
# Stitch Fix





# AI at Stitch Fix

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

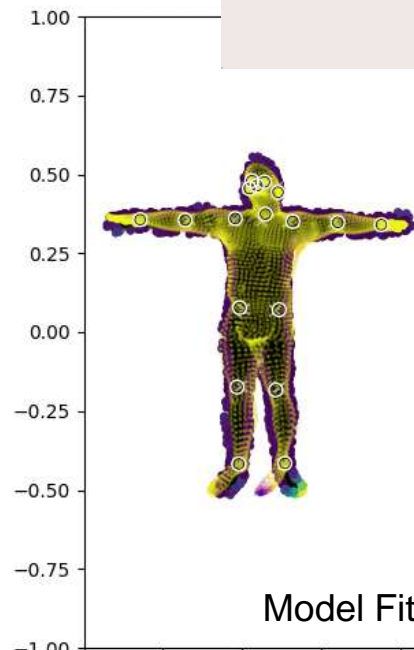


Capture

Joint  
Annotation



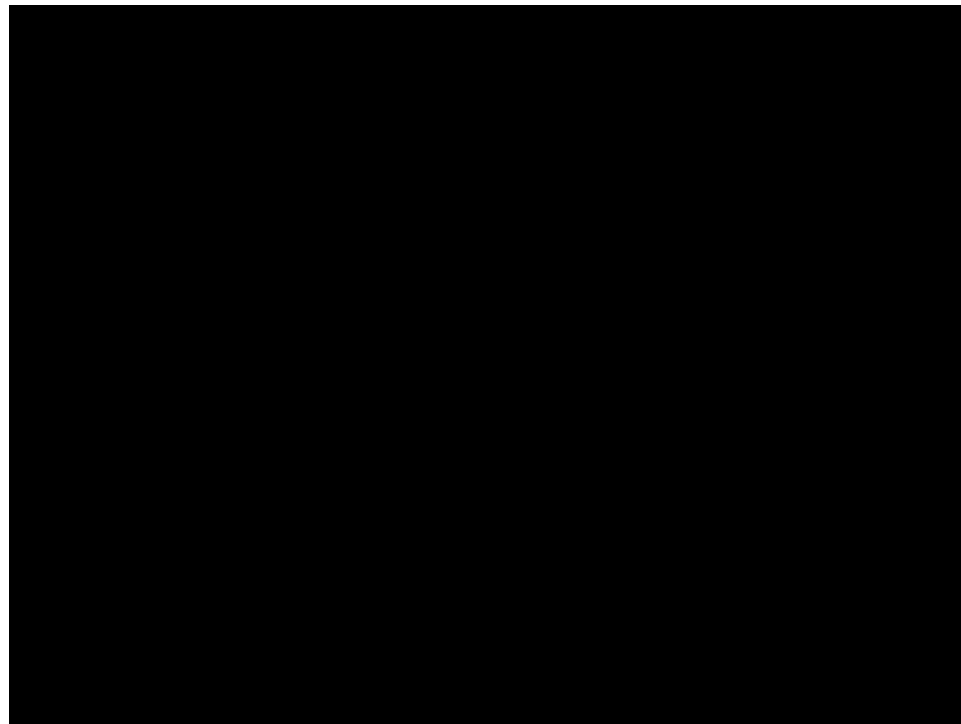
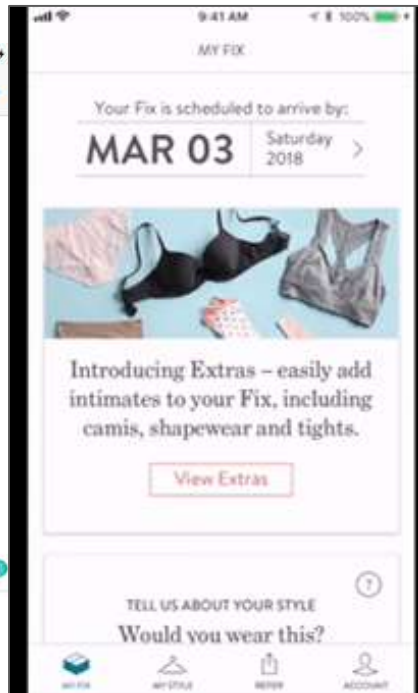
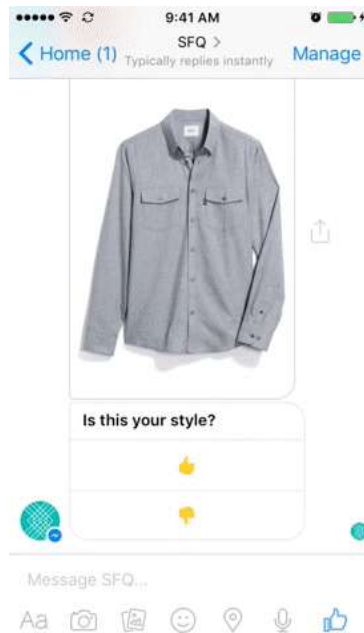
Segmentation



Model Fitting

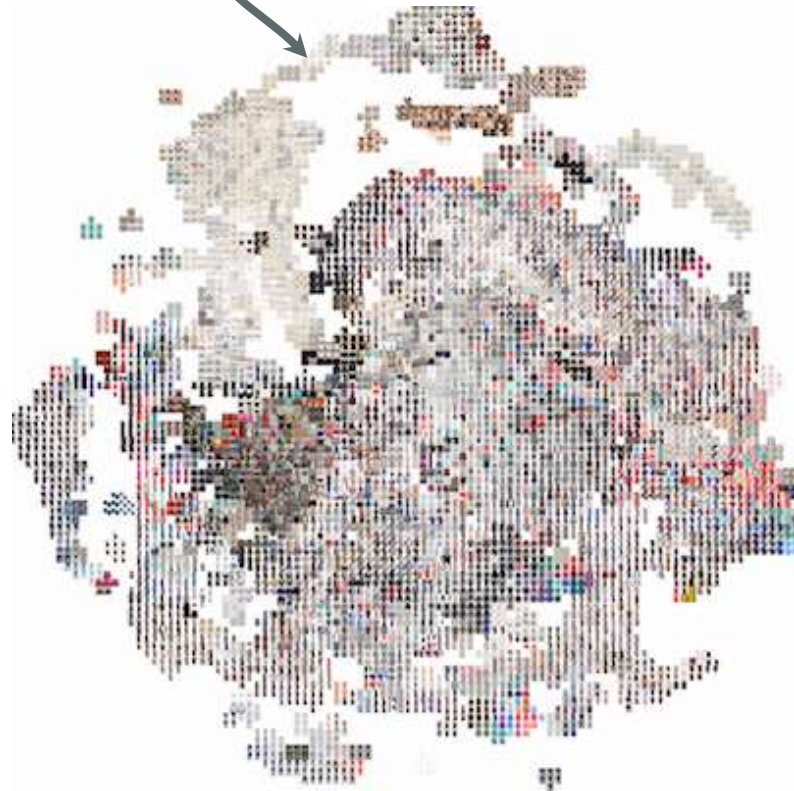
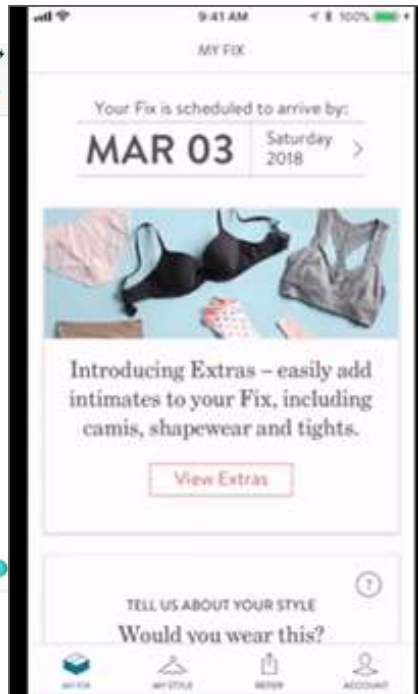
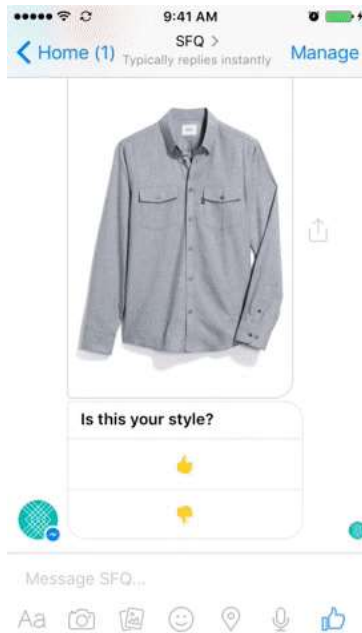
# “Latent” Style Space

Matrix  
Factorization



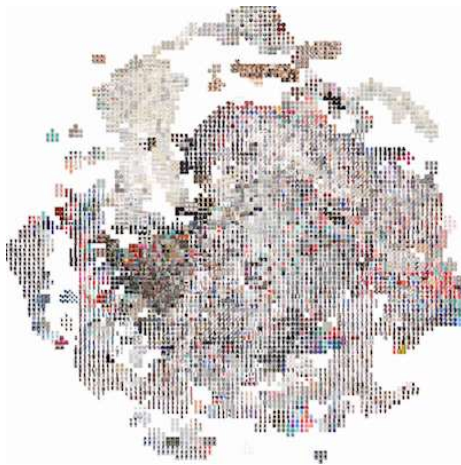
# “Latent” Style Space

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!*

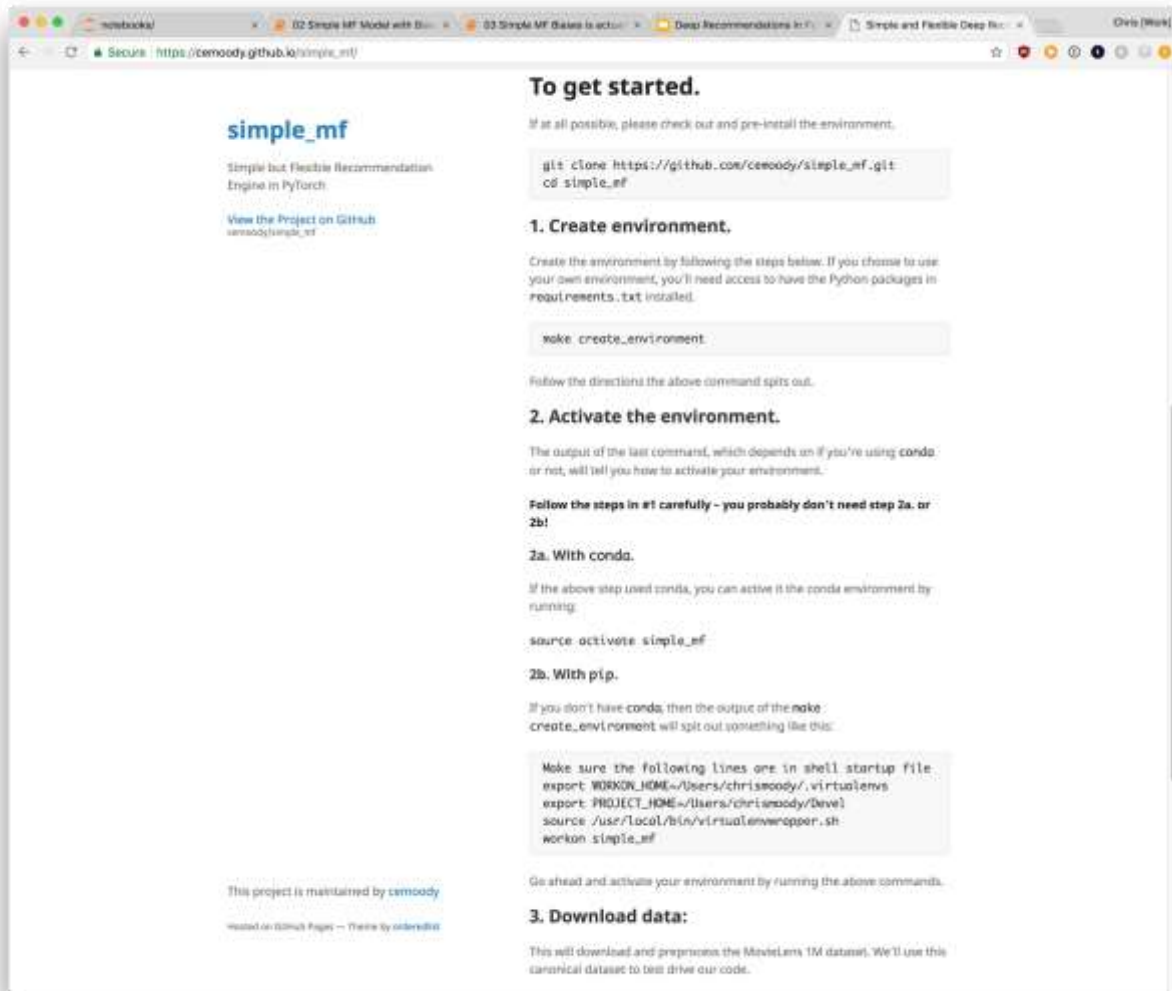
1. **Custom** models respect your heterogeneous and unique system

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

# Setup

Follow along with instructions here:

[cemoodys.github.io/simple\\_mf](https://cemoodys.github.io/simple_mf)



The screenshot shows the GitHub repository page for `simple_mf` by `cemoodys`. The page is titled "simple\_mf" and describes it as a "Simple but Flexible Recommendation Engine in PyTorch". It provides instructions on how to get started, including cloning the repository, creating and activating the environment, and downloading data.

**simple\_mf**  
Simple but Flexible Recommendation Engine in PyTorch  
[View the Project on GitHub](#)  
[cemoodys/simple\\_mf](#)

**To get started.**  
If at all possible, please check out and pre-install the environment.

```
git clone https://github.com/cemoodys/simple_mf.git
cd simple_mf
```

**1. Create environment.**  
Create the environment by following the steps below. If you choose to use your own environment, you'll need access to have the Python packages in `requirements.txt` installed.

```
make create_environment
```

Follow the directions the above command spits out.

**2. Activate the environment.**  
The output of the last command, which depends on if you're using `conda` or not, will tell you how to activate your environment.

**Follow the steps in it carefully - you probably don't need step 2a. or 2b!**

**2a. With conda.**  
If the above step used `conda`, you can active it the `conda` environment by running:

```
source activate simple_mf
```

**2b. With pip.**  
If you don't have `conda`, then the output of the `make create_environment` will spit out something like this:

```
Make sure the following lines are in shell startup file
export WORKON_HOME=~/.virtualenvs
export PROJECT_HOME=~/.virtualenvs/Devel
source /usr/local/bin/virtualenvwrapper.sh
workon simple_mf
```

Go ahead and activate your environment by running the above commands.

**3. Download data:**  
This will download and preprocess the MovieLens 1M dataset. We'll use this canonical dataset to test drive our code.

This project is maintained by `cemoodys`  
Hosted on GitHub Pages — Theme by `oceanicbird`

# Why PyTorch

# functional-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)
```

# 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)
```

PYTORCH

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

K

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

# 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)
```

PYTORCH

K

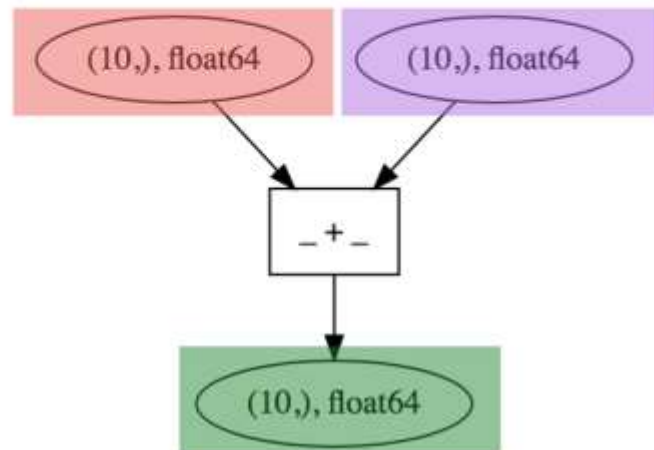
Keras

# 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
```

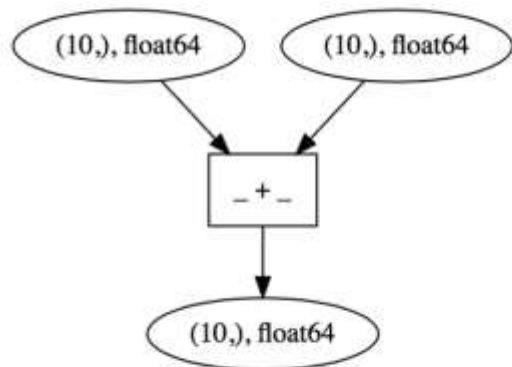




# theano

TensorFlow

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

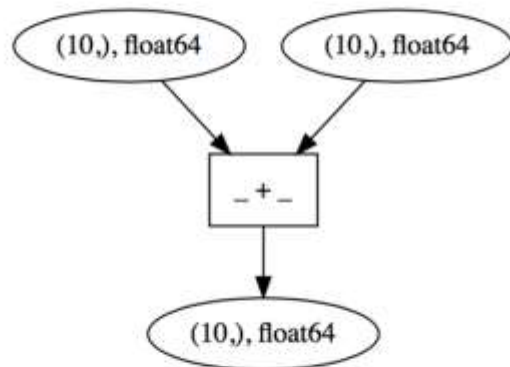


```
In [47]: loss
Out[47]: theano.tensor.var.TensorVariable
```

**symbolic variable**

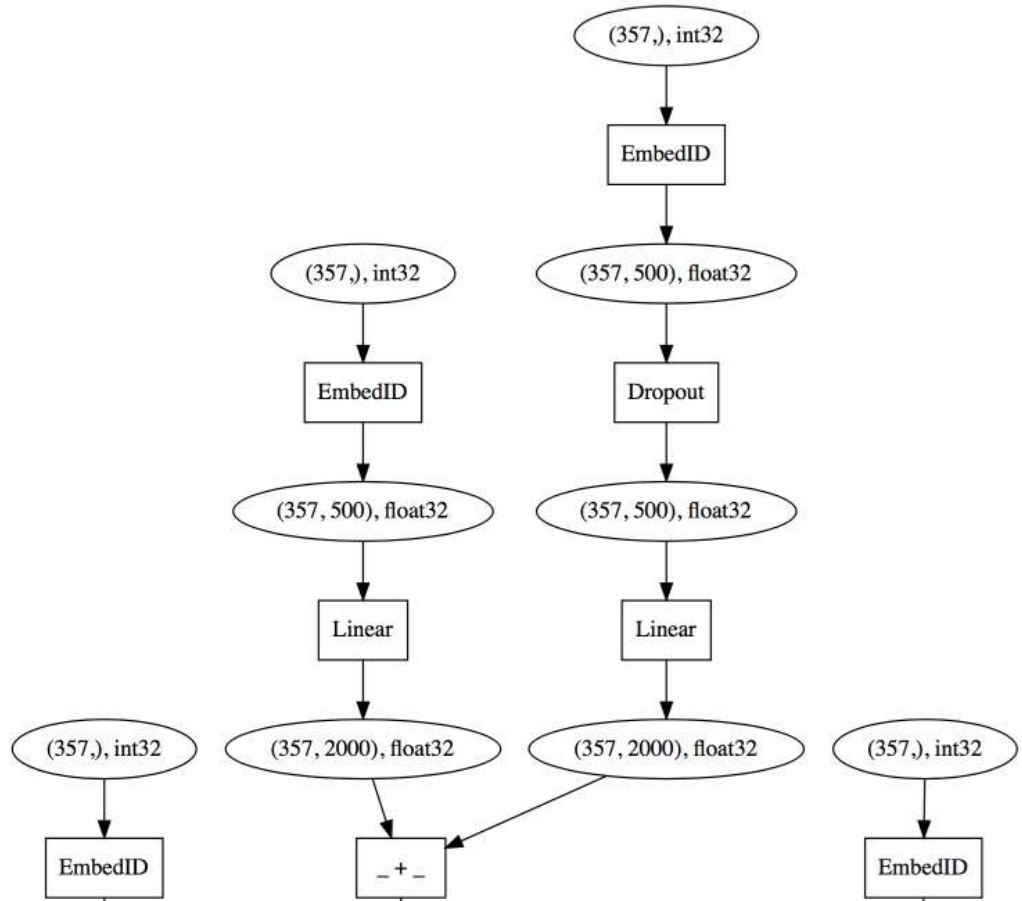
# PYTORCH

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



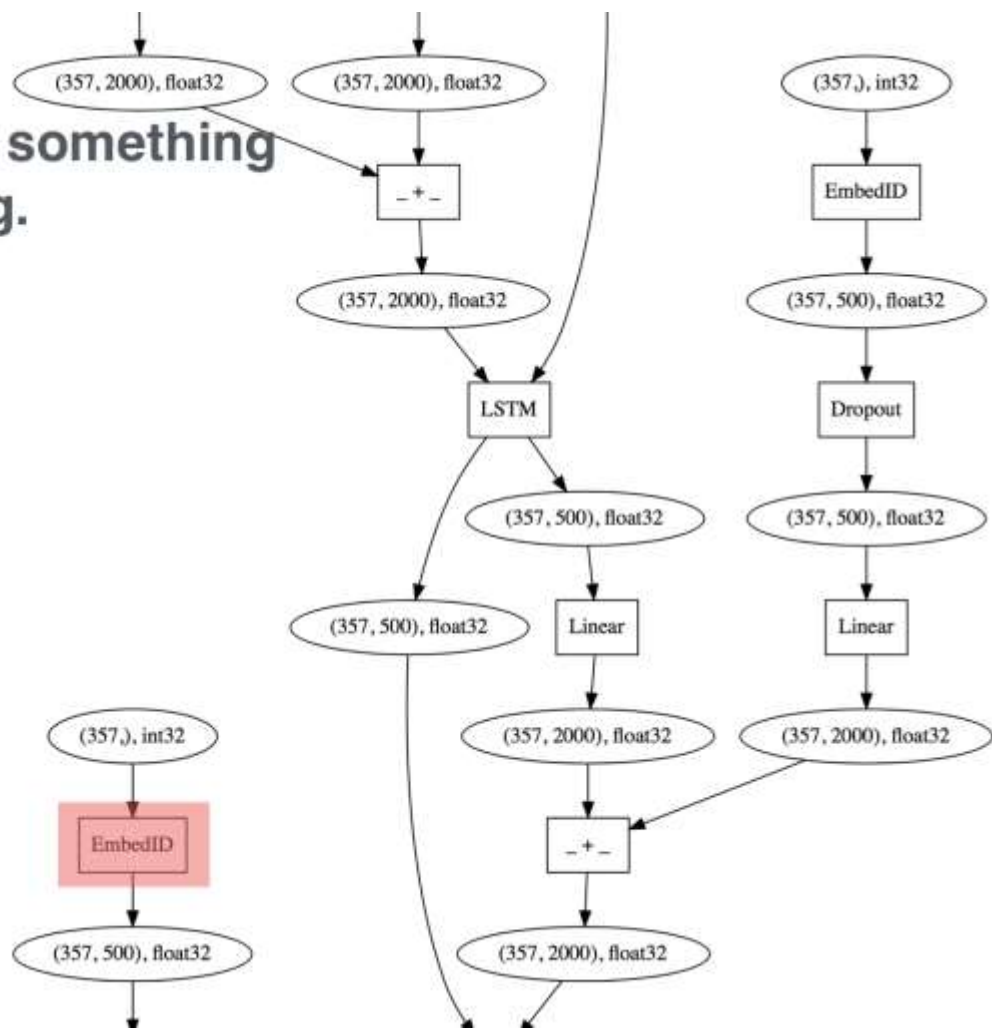
```
In [47]: loss.data
Out[47]: array([ 2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.,  2.])
```

**symbolic + numeric variable**



This gets *very* deep.

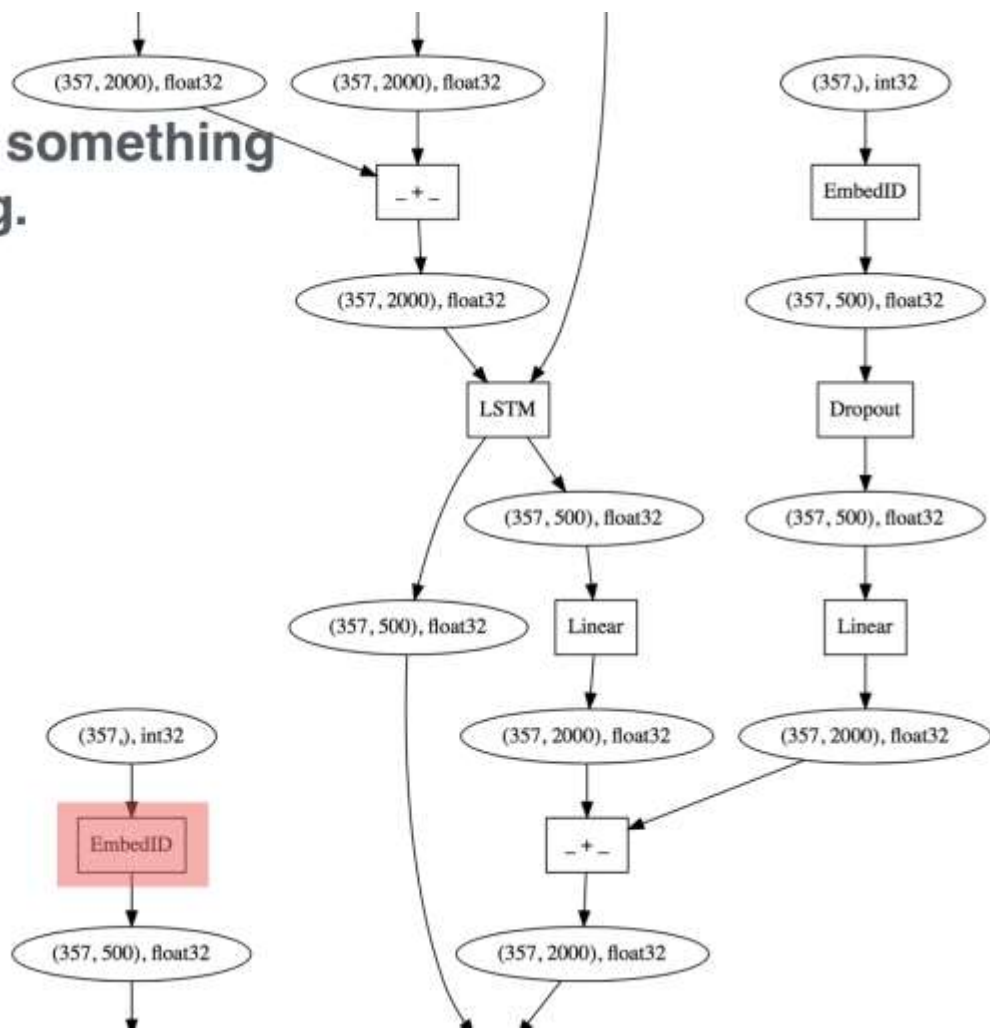
...and then something goes wrong.





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

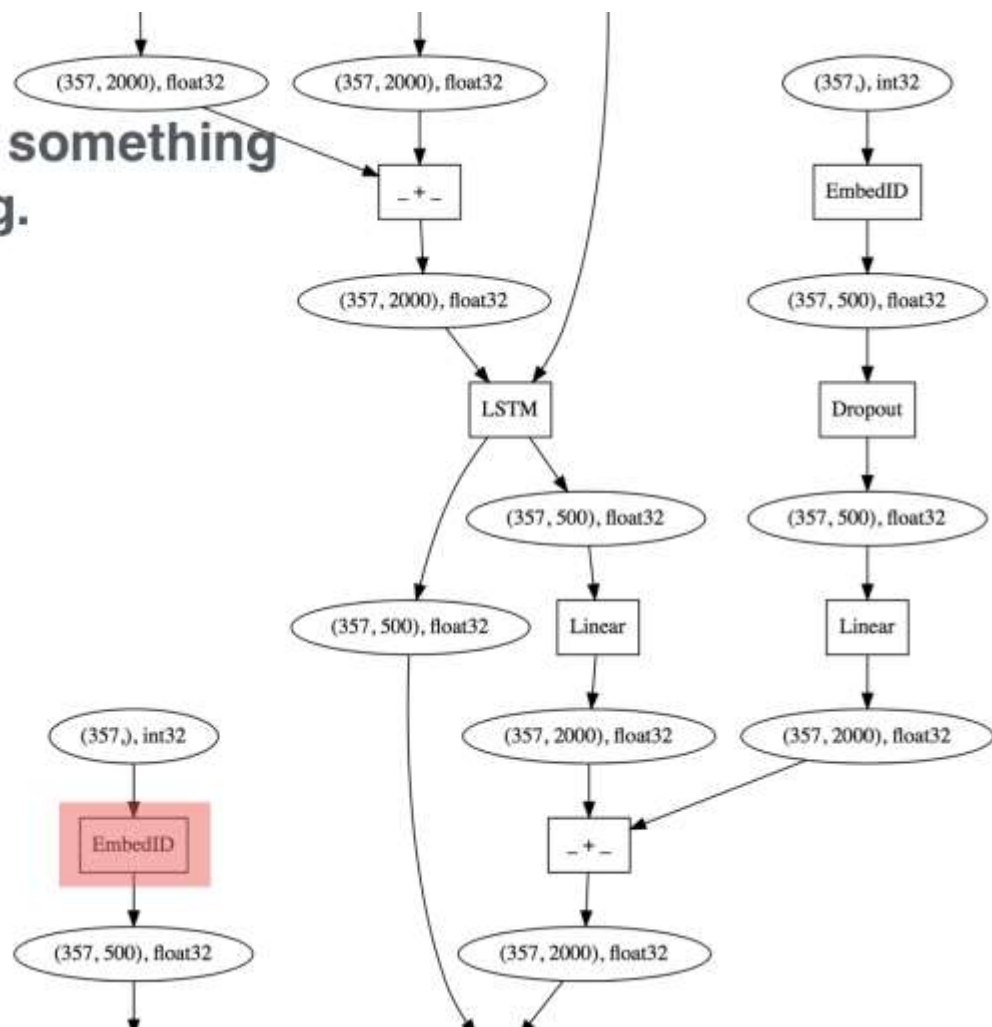
...and then something  
goes wrong.



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

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

...and then something  
goes wrong.

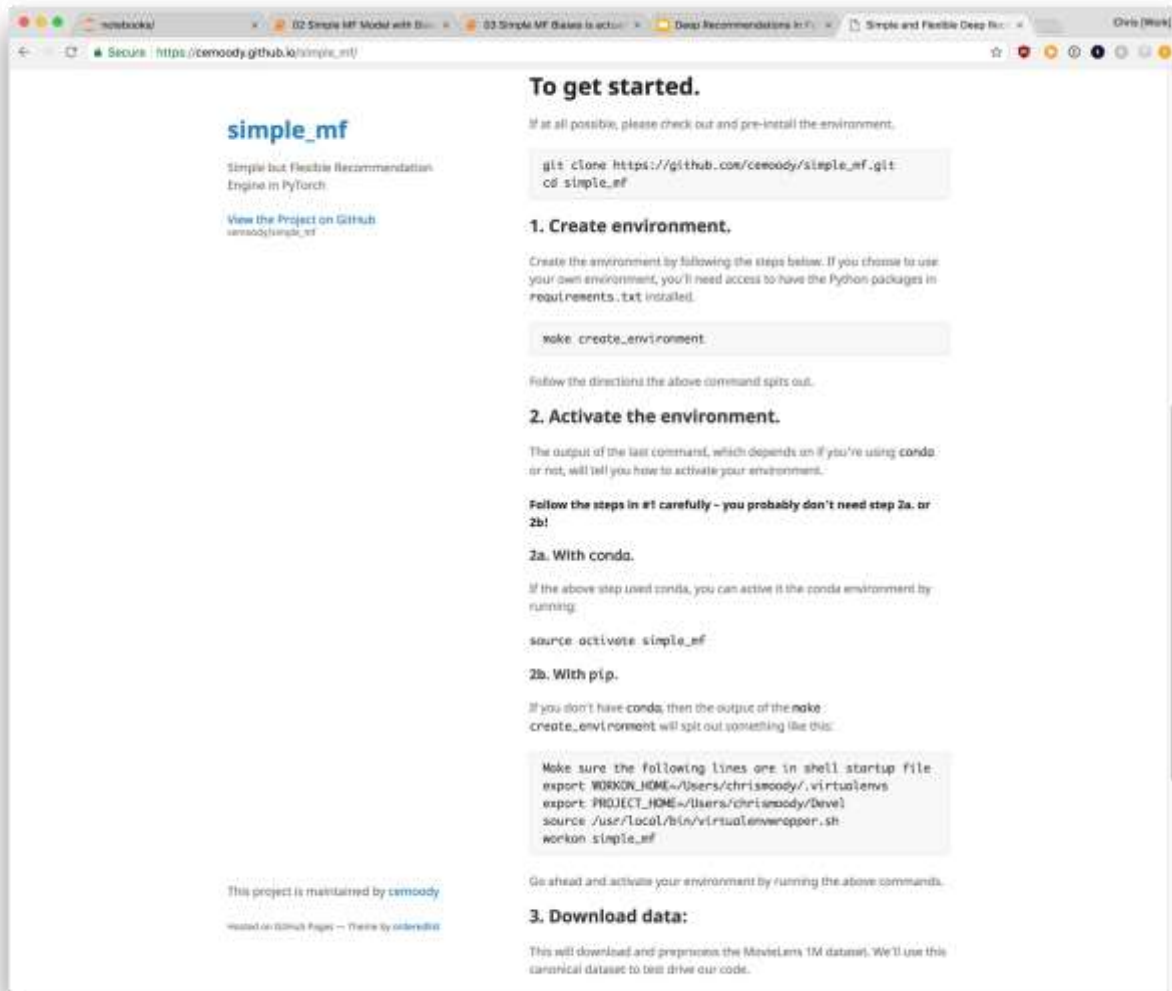


# Setup

# Setup

Follow along with instructions here:

[cemoodys.github.io/simple\\_mf](https://cemoodys.github.io/simple_mf)



The screenshot shows the GitHub repository page for `simple_mf` by `cemoodys`. The page is titled "simple\_mf" and describes it as a "Simple but Flexible Recommendation Engine in PyTorch". It includes a link to "View the Project on GitHub" and a note that the project is maintained by `cemoodys`.

**To get started.**

If at all possible, please check out and pre-install the environment.

```
git clone https://github.com/cemoodys/simple_mf.git
cd simple_mf
```

**1. Create environment.**

Create the environment by following the steps below. If you choose to use your own environment, you'll need access to have the Python packages in `requirements.txt` installed.

```
make create_environment
```

Follow the directions the above command spits out.

**2. Activate the environment.**

The output of the last command, which depends on if you're using `conda` or not, will tell you how to activate your environment.

**Follow the steps in it carefully - you probably don't need step 2a. or 2b!**

**2a. With conda.**

If the above step used `conda`, you can active it the `conda` environment by running:

```
source activate simple_mf
```

**2b. With pip.**

If you don't have `conda`, then the output of the `make create_environment` will spit out something like this:

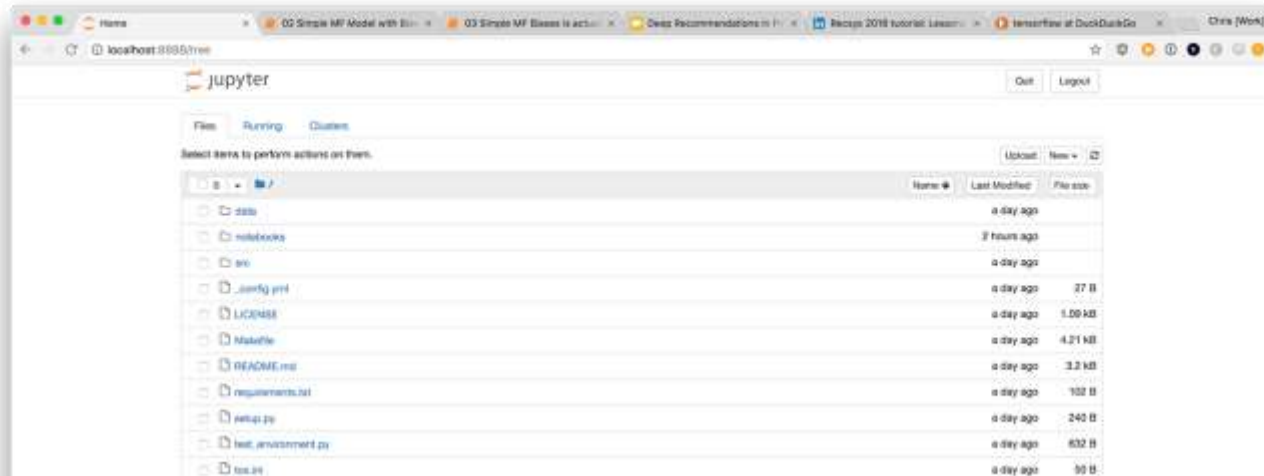
```
Make sure the following lines are in shell startup file
export WORKON_HOME=~/.virtualenvs
export PROJECT_HOME=~/.virtualenvs/Devel
source /usr/local/bin/virtualenvwrapper.sh
workon simple_mf
```

Go ahead and activate your environment by running the above commands.

**3. Download data:**

This will download and preprocess the MovieLens 1M dataset. We'll use this canonical dataset to test drive our code.

## Start Jupyter Notebook



```
chrismoodymBP15-16-500-TV0PW ~-> bash
bash-3.2$ source ~/.virtualenvs/simple_mf/bin/activate
(simple_mf) bash-3.2$ jupyter notebook
[I 19:08:09.228 NotebookApp] The port 8888 is already in use, trying another port.
[I 19:08:09.269 NotebookApp] Serving notebooks from local directory: /Users/chrismoodym
[I 19:08:09.269 NotebookApp] The Jupyter Notebook is running at:
[I 19:08:09.269 NotebookApp] http://localhost:8889/?token=92a5c5cbb36e641f6c9b22fbbba340b5e887165bc91e2da
[I 19:08:09.269 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 19:08:09.271 NotebookApp]
```

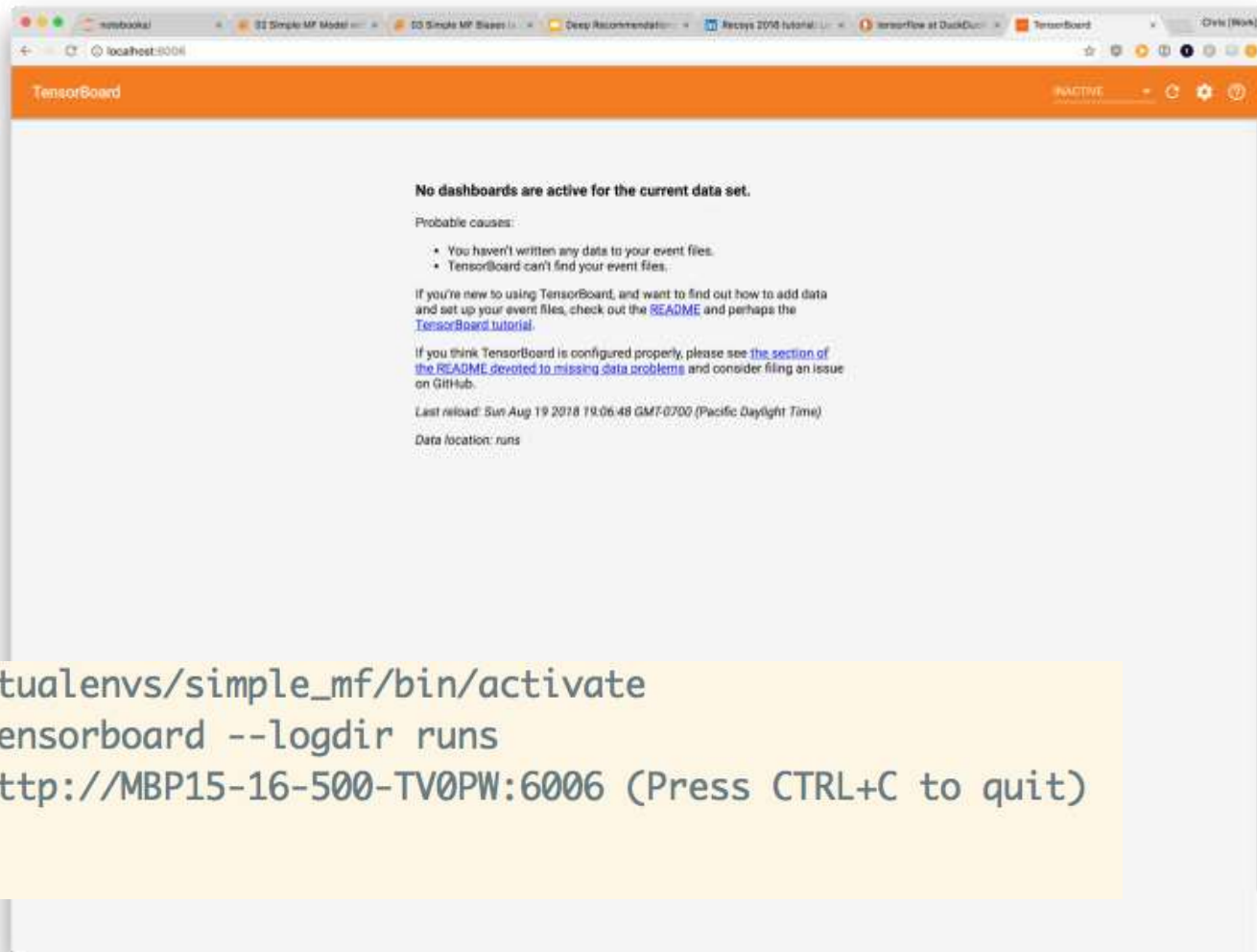
Copy/paste this URL into your browser when you connect for the first time,  
to login with a token:

<http://localhost:8889/?token=92a5c5cbb36e641f6c9b22fbbba340b5e887165bc91e2da>

```
[I 19:08:09.588 NotebookApp] Accepting one-time-token-authenticated connection from ::1
```

## 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)
```

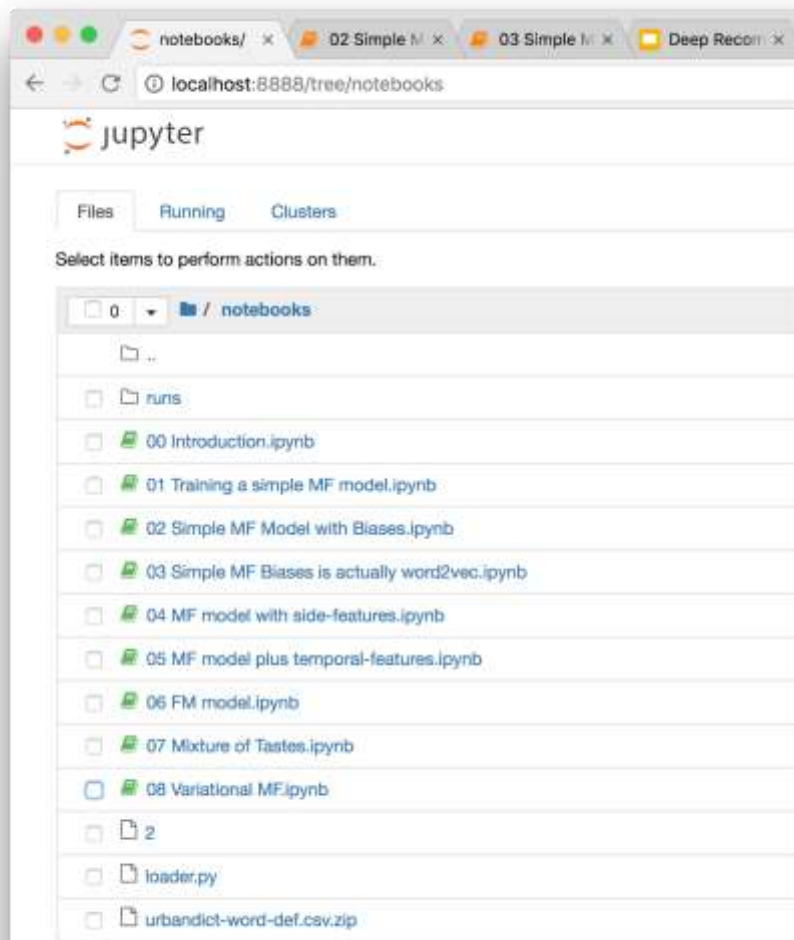


There are eight Jupyter Notebooks in this tutorial.

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.



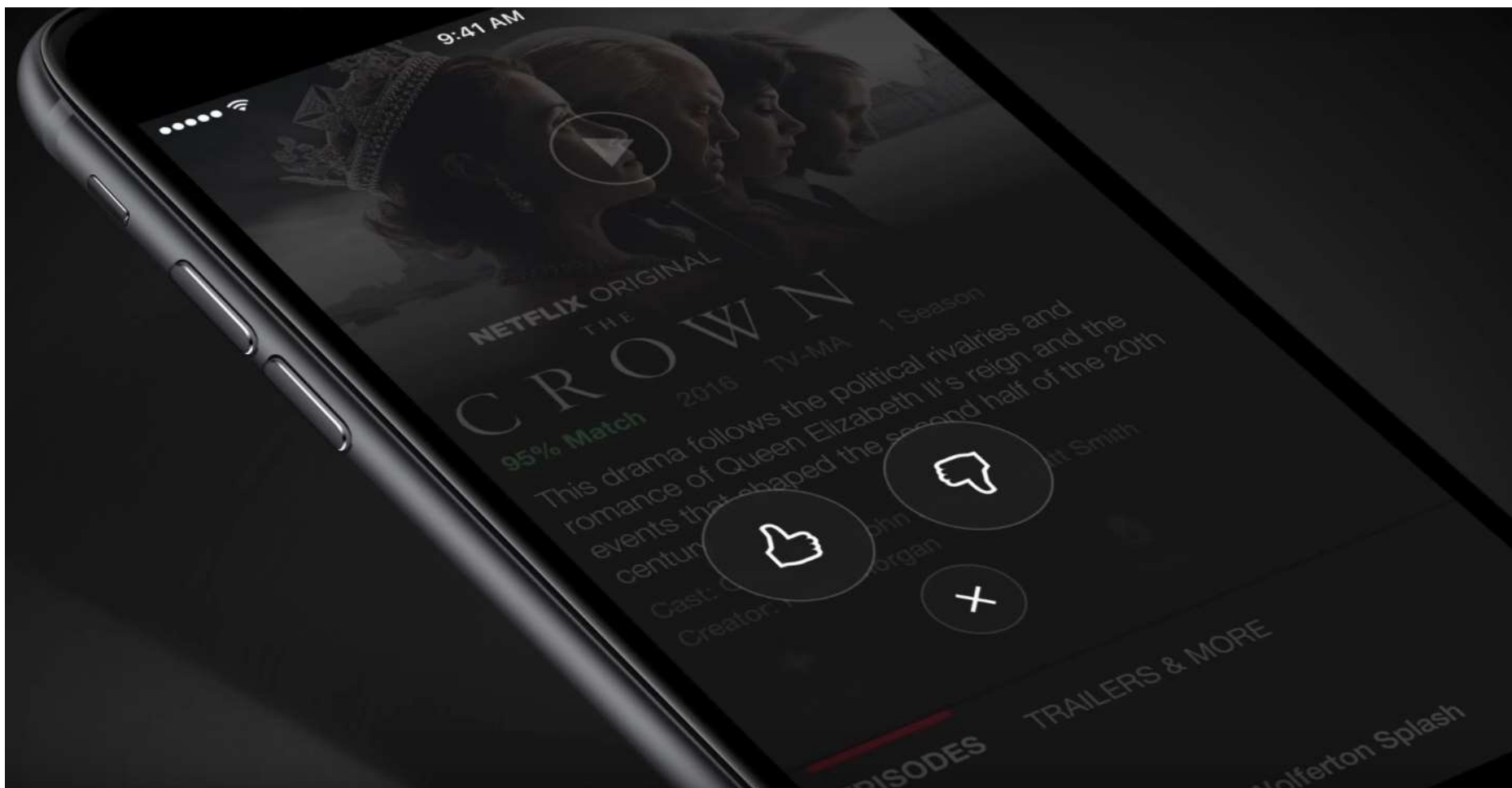


# Matrix Factorization Fundamentals

**Notebooks we'll be using:**

01 Training a simple MF model.ipynb

02 Simple MF Model with Biases.ipynb



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?

 MY FIX

 MY STYLE

 REVER

 ACCOUNT



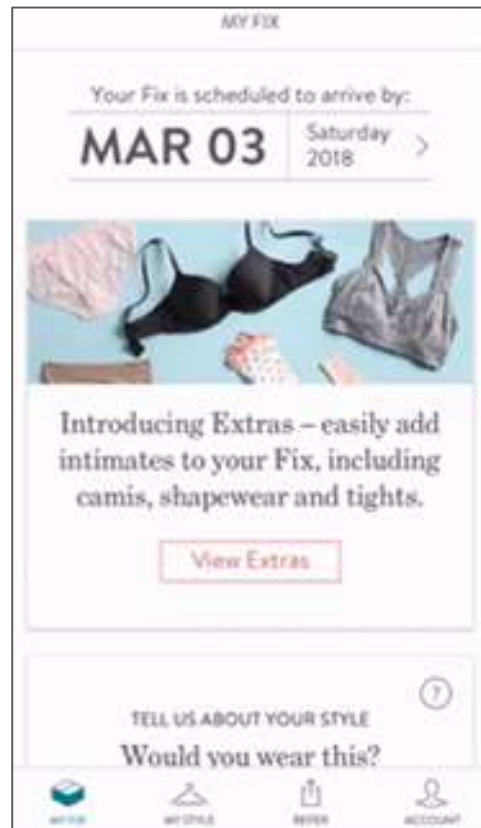
+1

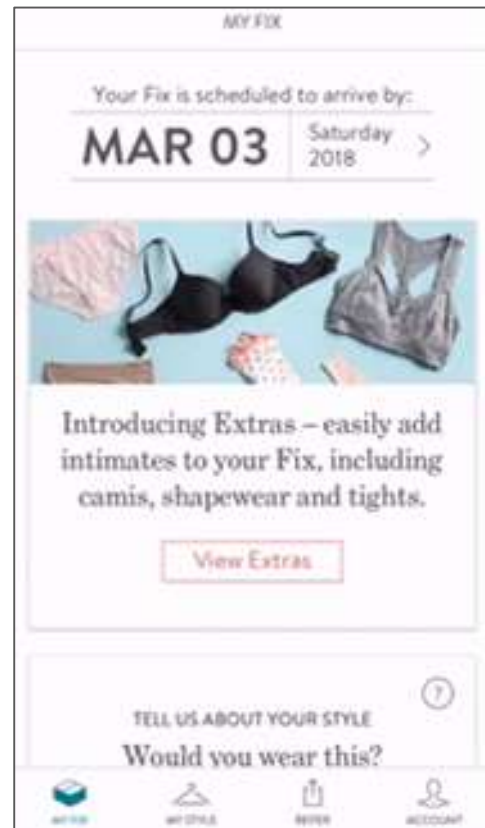


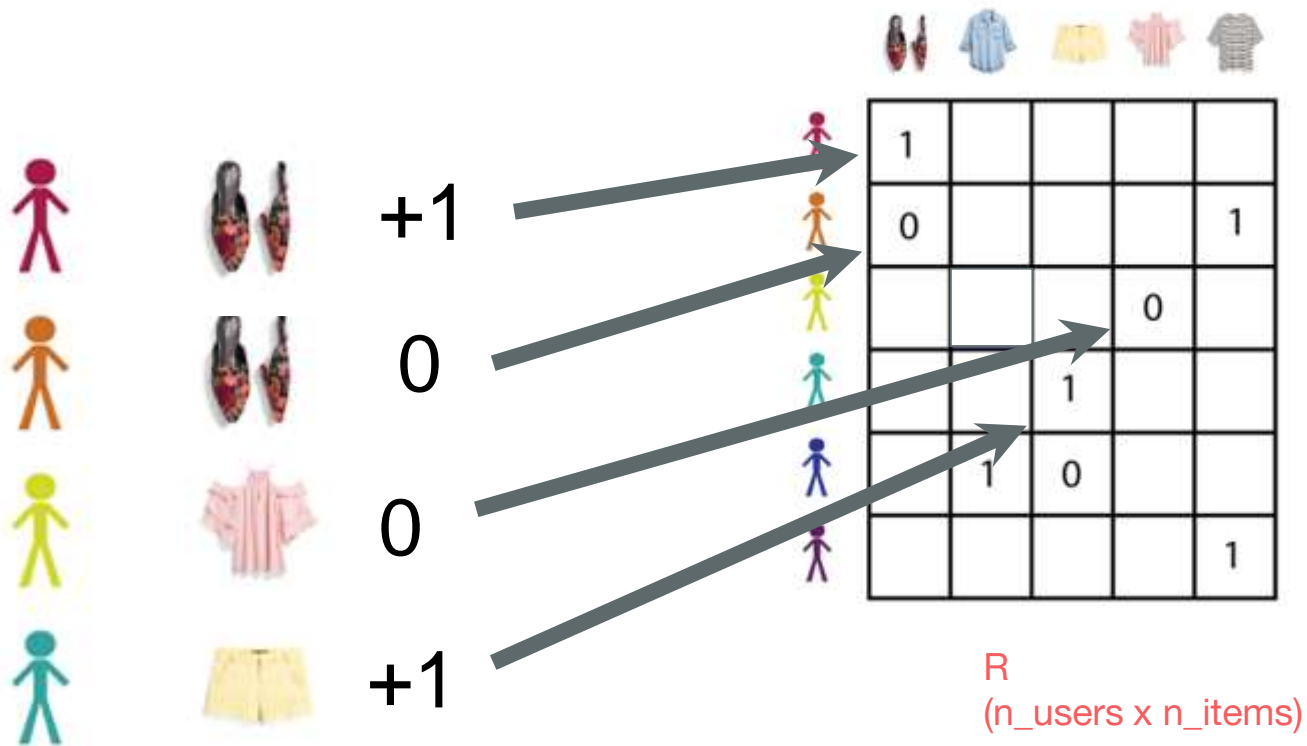
0



0
















# R

The *ratings matrix*  
( $n_{\text{users}} \times n_{\text{items}}$ )



	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*  
( $n_{\text{users}}, n_{\text{items}}$ )



	1				
	0				1
				0	
			1		
		1	0		
					1

# P

The *user matrix*  
( $n_{\text{users}}, k$ )

~

# Q

The *item matrix*  
( $n_{\text{item}}, k$ )

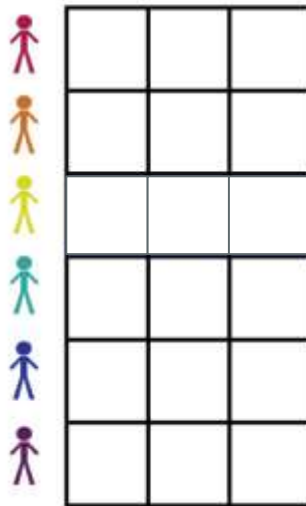



**Much smaller!**  
(millions or 100ks of rows)

**Compact & dense!**  
No zeroes, efficient storage.

**P**

The *user matrix*  
( $n_{\text{users}}, k$ )



A diagram of the user matrix. It consists of a 6x3 grid of empty squares. To the left of each row is a small stick figure icon of a different color: red, orange, yellow, green, blue, and purple from top to bottom.

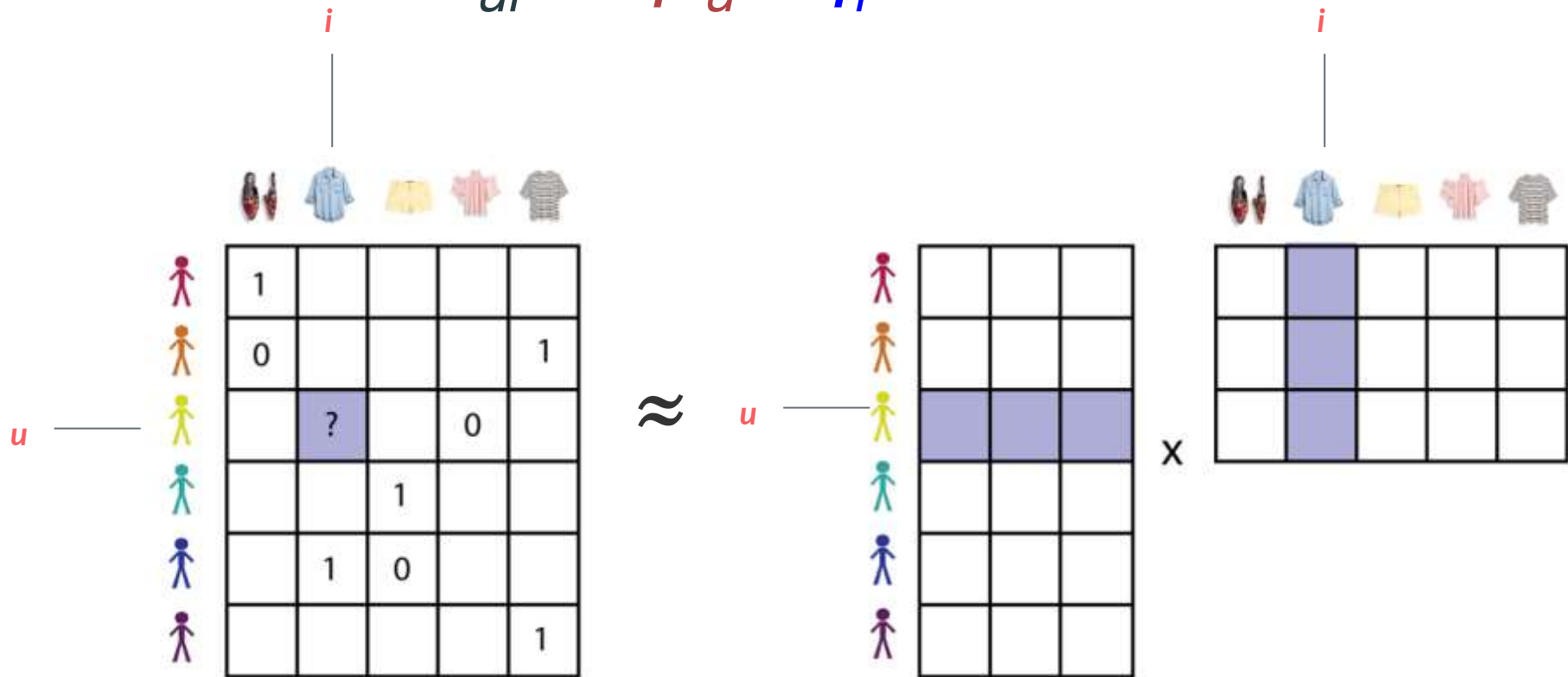

**Q**

The *item matrix*  
( $n_{\text{item}}, k$ )



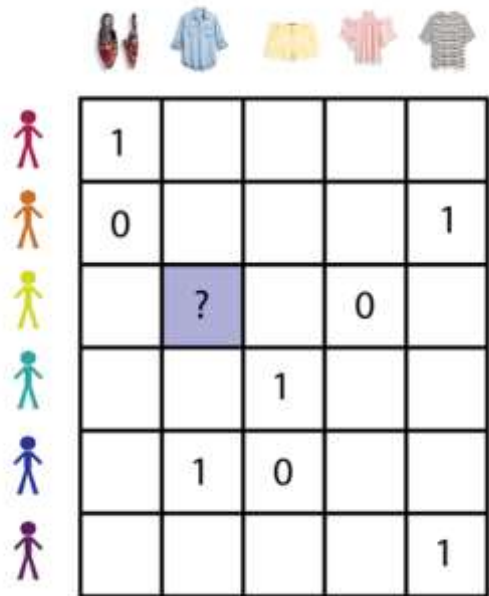
A diagram of the item matrix. It consists of a 3x5 grid of empty squares. Above each column is a small icon of a different item: a pair of red shoes, a blue shirt, a yellow skirt, a pink shirt, and a grey t-shirt.


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



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

A single +1 or 0 rating

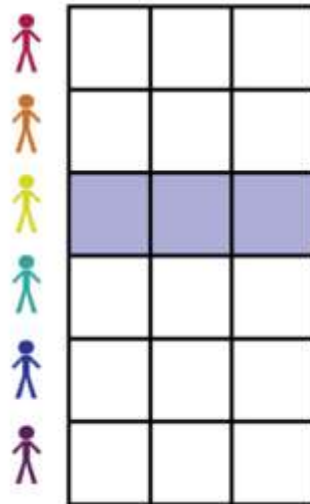


	1				
	0				1
		?		0	
			1		
		1	0		
					1

known

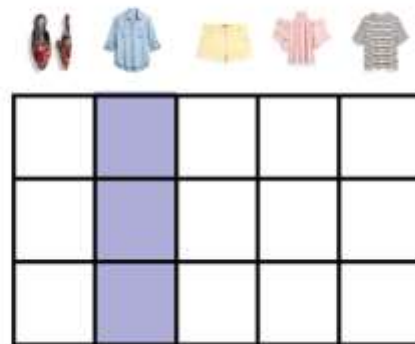
$\approx$

User vector




to be estimated

Item vector







x

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

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

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*.

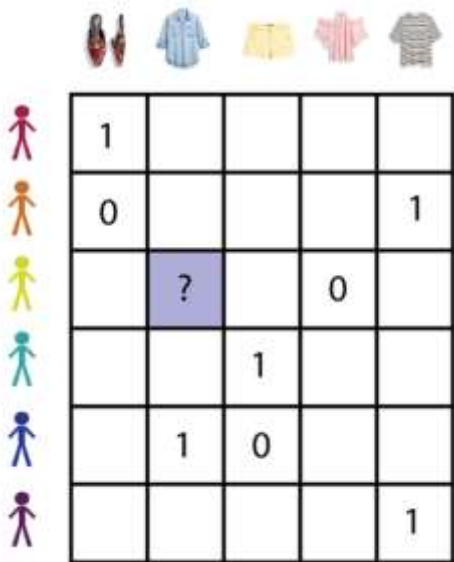
$$R_{ui} = \boxed{b + \omega_u + \mu_i} + p_u \cdot q_i$$

Model +  
global bias  
user biases  
item biases

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

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

A single +1 or 0 rating



	1				
	0				1
		?		0	
			1		
		1	0		
					1

=

User bias

One number  
per user

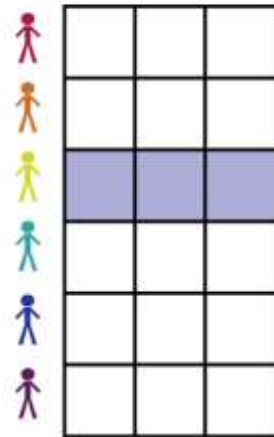



Item bias

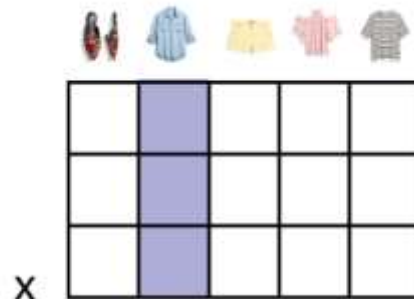
One number  
per item




User vector



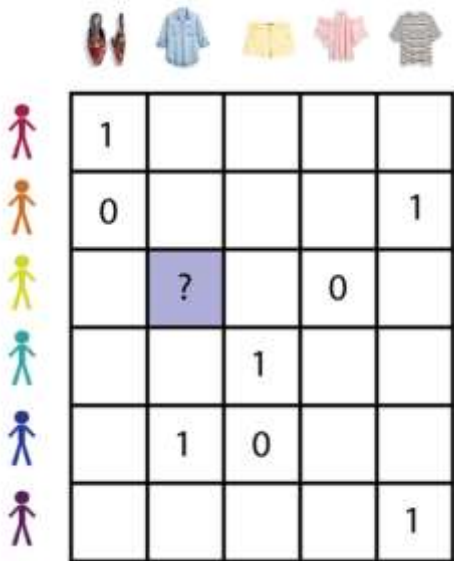

Item vector




x

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

A single +1 or 0 rating



	1				
	0				1
		?		0	
			1		
		1	0		
					1

=

User bias

One number  
per user

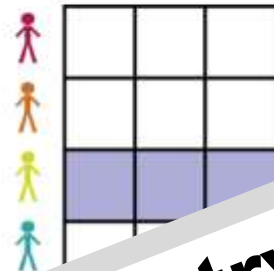


Item bias

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.

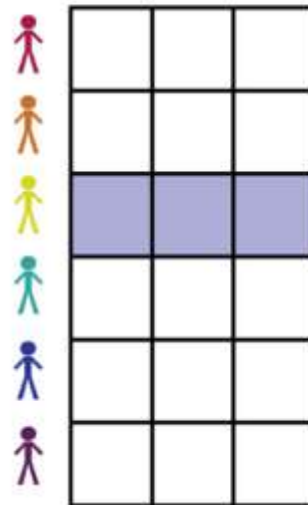
Stitch Fix vectors stand for **clothing style**.

Movielens vectors yield **movie genres**.

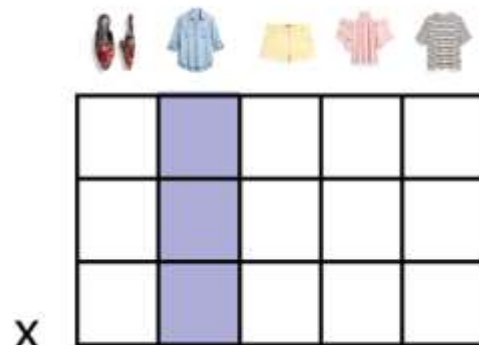
At Spotify, they represent latent **musical tastes**.

Amazon they represent latent **categories**.

User vector



Item vector



X

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





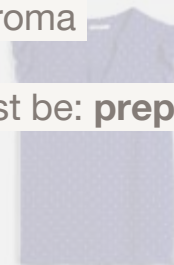
tile-printed,  
paisley,  
bell-sleeved,  
fringed,  
tasseled,  
maxi



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....

... we have many more.

What do each of those look like?

+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

Internally, we use 18 dimensions.

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

[Engineering](#)[Algorithms](#)[Careers](#)[Blog](#)

# Understanding Latent Style



ERIN BOYLE AND JANA BECK

June 28, 2018 - San Francisco, CA



Tweet this post!



Post on LinkedIn

At Stitch Fix, we approach fashion recommendations with a humans-in-the-loop philosophy. In some cases this means coupling a machine learning model with human oversight and

**Let's try it!**  
Check out tensorboard

For more search:

[“Understanding Latent Style”](#)

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

# 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**

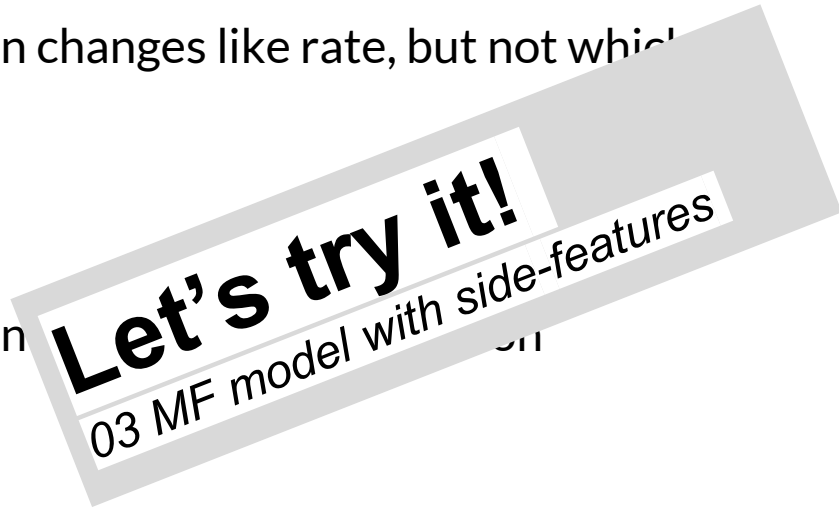
- Choose this if you think occupation changes like rate, but not which movies

- 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”



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

What about when features change in time?

You've seen **biases**.

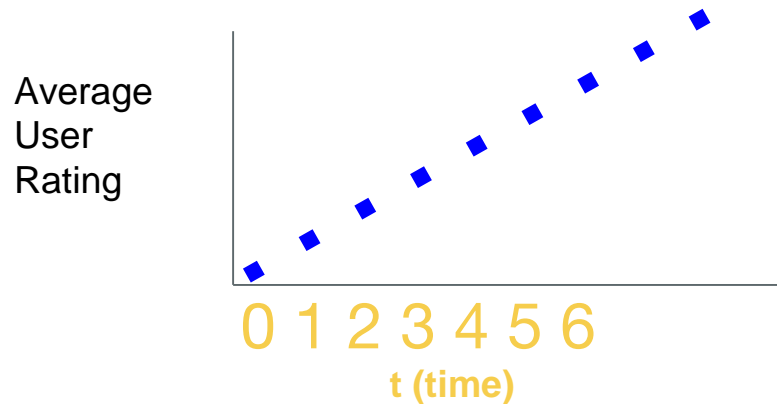
You've seen **user-item interactions**.

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

What does a latent temporal vector encode?

Here's a toy example.

$$m_0 = \text{[User Icon]} \begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$$

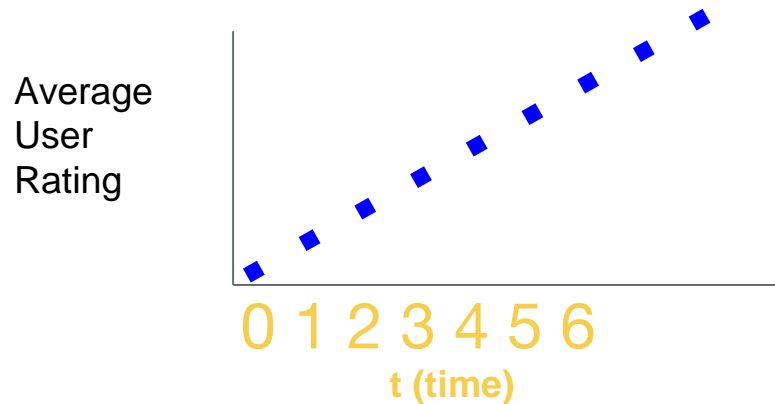


⋮

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

What does a latent temporal vector encode?

Here's a toy example.



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

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

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

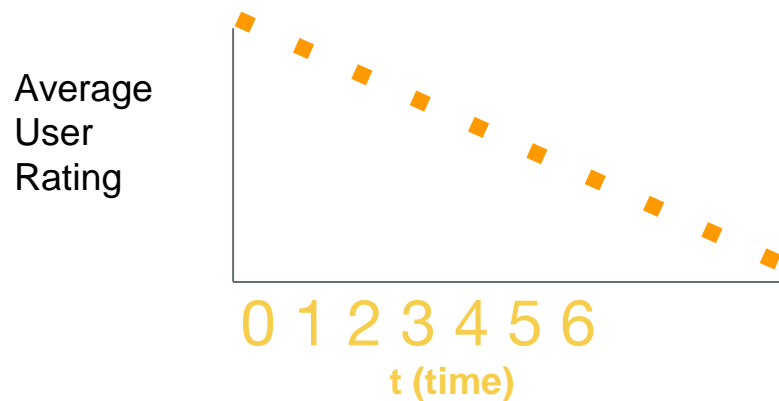
$$\vdots$$

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

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

What does a latent temporal vector encode?

Here's a toy example.



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

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

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

$$\vdots$$

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



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

What does a latent temporal vector encode?

A user mixture of time-series.

$$m_0 = \text{User 0} \begin{bmatrix} 0.0 & 1.0 \end{bmatrix}$$

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

$$m_1 = \text{User 1} \begin{bmatrix} 1.0 & 0.0 \end{bmatrix}$$

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

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

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

$$\vdots$$

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

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

We can enforce that nearby times have similar components.

Enforce that:

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

should be small.

$$n_0 = \text{calendar icon} \begin{array}{|c|c|} \hline 1.0 & 0.0 \\ \hline \end{array}$$

$$n_1 = \text{calendar icon} \begin{array}{|c|c|} \hline 1.0 & 0.0 \\ \hline \end{array}$$

$$n_t = \text{calendar icon} \begin{array}{|c|c|} \hline 1-t & t \\ \hline \end{array}$$

**Let's try it!**

04 MF model plus temporal-features

# 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*

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

## 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*

*w*  
↓

$$X[\textcolor{green}{c}, \textcolor{brown}{w}] += 1$$

## Co-occurrence modeling

“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery accents”



*c*



*w*

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

## Co-occurrence modeling

“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery accents”

↑  
*c*

↑  
*w*

$$X[\textcolor{green}{c}, \textcolor{brown}{w}] += 1$$

## Co-occurrence modeling

“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery accents”

↑  
*c*

↑  
*w*

$$X[\textcolor{green}{c}, \textcolor{brown}{w}] += 1$$

## Co-occurrence modeling

“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery accents”

↑  
*c*

↑  
*w*

$$X[\textcolor{teal}{c}, \textcolor{brown}{w}] += 1$$

## Co-occurrence modeling

*w*      *w*      *w*      *w*      *w*  
↓      ↓      ↓      ↓      ↓  
“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery accents”

↑      ↑      ↑      ↑      ↑  
*c*      *w*      *w*      *w*      *w*

$$X[\textcolor{green}{c}, \textcolor{brown}{w}] = \text{count}$$

## Co-occurrence modeling

*w*      *w*      *w*      *w*      *w*  
↓  
“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery accents”  
↑

*w*      *c*      *w*      *w*      *w*  
↑

$$X[\textit{c}, \textit{w}] = \text{count}$$

## Co-occurrence modeling

*w*      *w*      *w*      *w*      *w*  
↓      ↓      ↓      ↓      ↓  
“ITEM\_92 think fabric wonderful rayon  
spandex like *lace* embroidery accents”

↑      ↑      ↑      ↑      ↑  
*w*      *w*      *c*      *w*      *w*

$$X[\textit{c}, \textit{w}] = \text{count}$$



## Co-occurrence modeling

*w*      *w*      *w*      *w*      *w*  
↓      ↓      ↓      ↓      ↓  
“ITEM\_92 think fabric wonderful rayon  
spandex like lace **embroidery** accents”

↑      ↑      ↑      ↑      ↑  
*w*      *w*      *w*      *c*      *w*

$$X[\textcolor{green}{c}, \textcolor{brown}{w}] = \text{count}$$

## Co-occurrence modeling

*w*      *w*      *w*      *w*      *w*  
↓      ↓      ↓      ↓      ↓  
“ITEM\_92 think fabric wonderful rayon  
spandex like lace embroidery **accents**”

↑      ↑      ↑      ↑      ↑  
*w*      *w*      *w*      *w*      *c*

$$X[\textcolor{green}{c}, \textcolor{brown}{w}] = \text{count}$$

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

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!

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

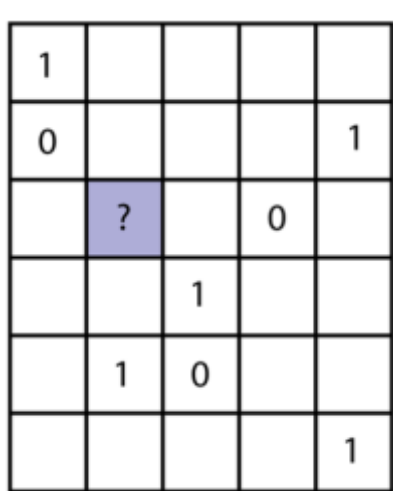
~~A single +1 or 0 rating~~  
Log Skipgram count

User bias  
token1 bias

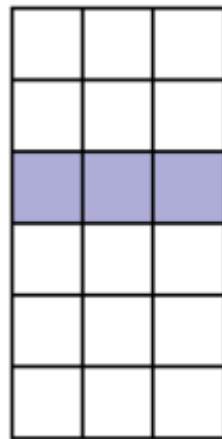
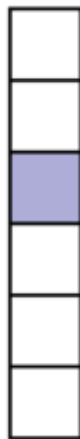
Item bias  
token2 bias

~~User vector~~  
token1 vector

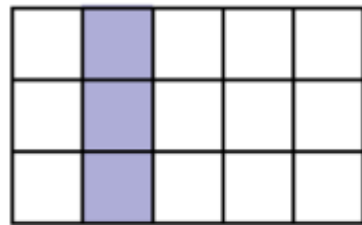
Item vector  
token2 vector



11



X



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

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

User bias  
token1 bias

(How frequent is this word?)

1				
0				1
	?		0	
		1		
	1	0		
				1

\_\_\_\_\_

\_\_\_\_\_


## Item bias

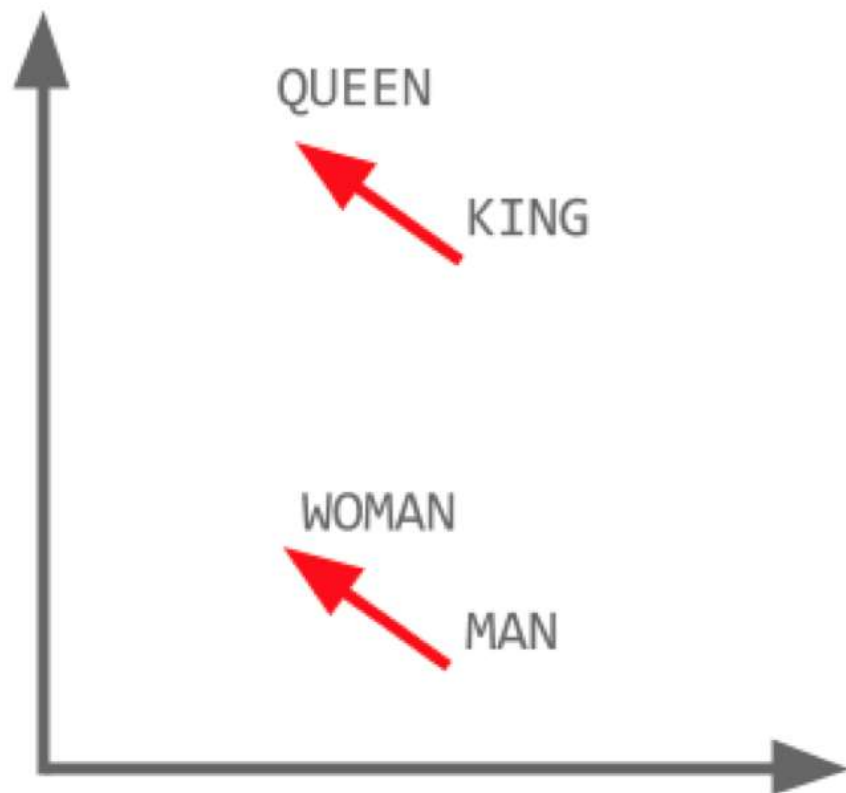
(How frequent is this word?)


~~User vector~~  
word1 vector

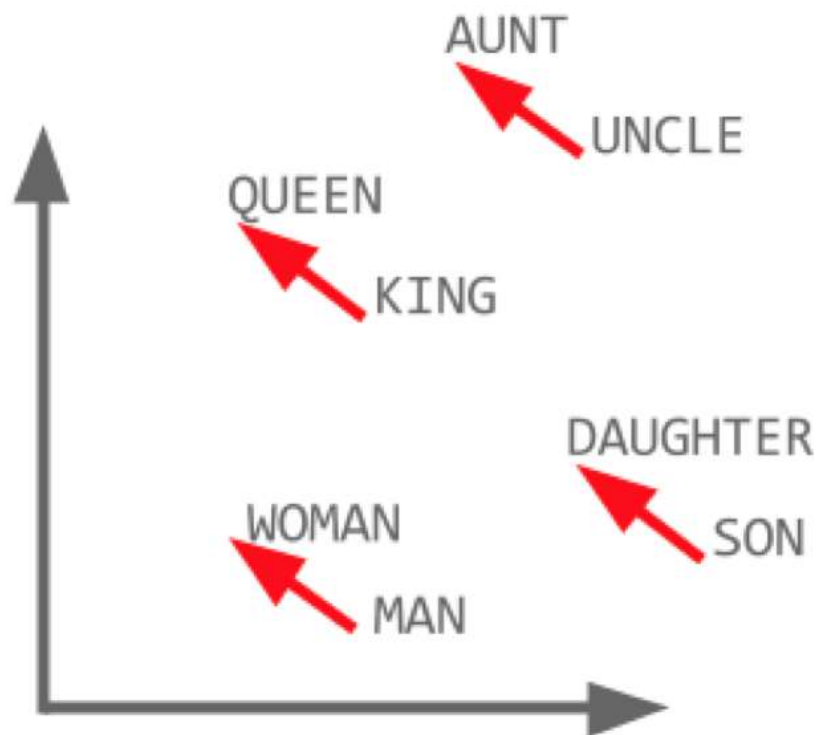
## Do word1 & word2 have a special interaction?


X


The **red direction** encodes gender



Which is consistent across all words

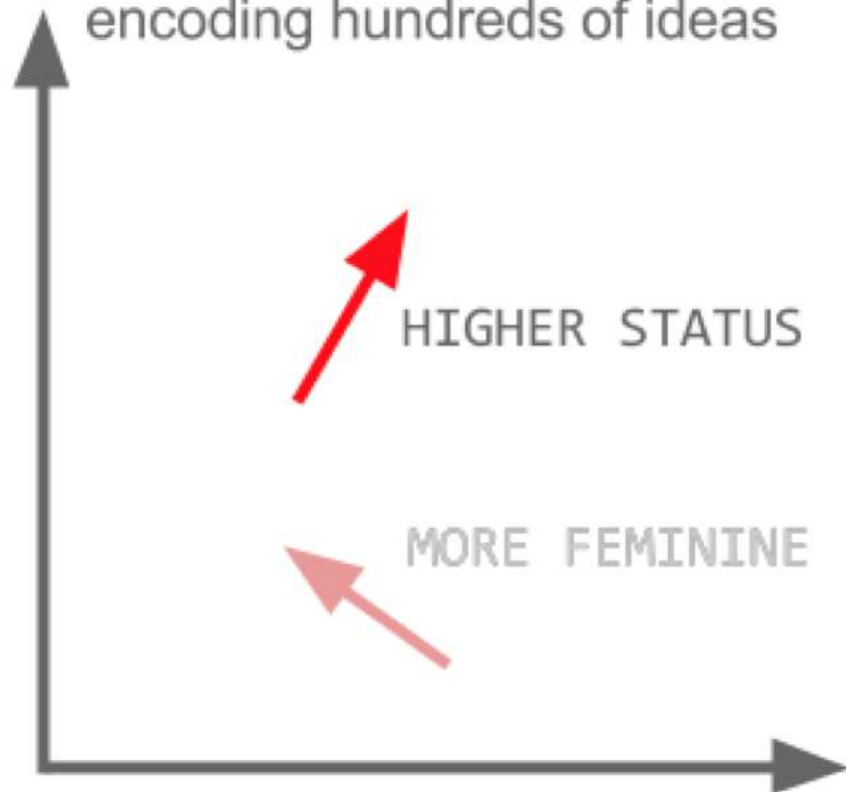


This **direction** always means **gender**

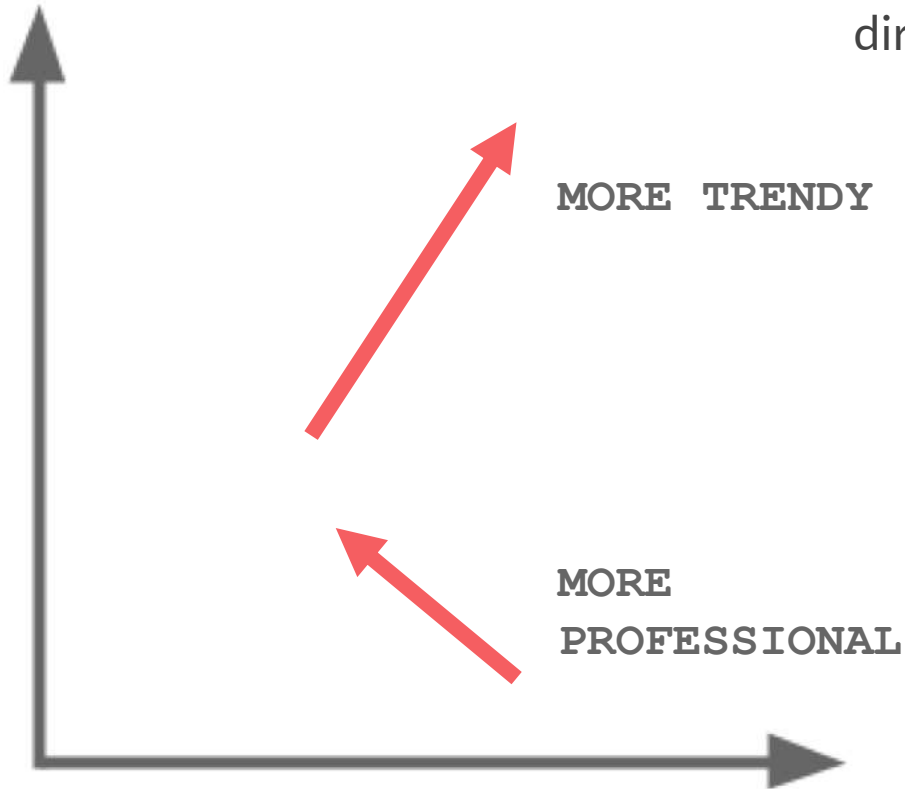




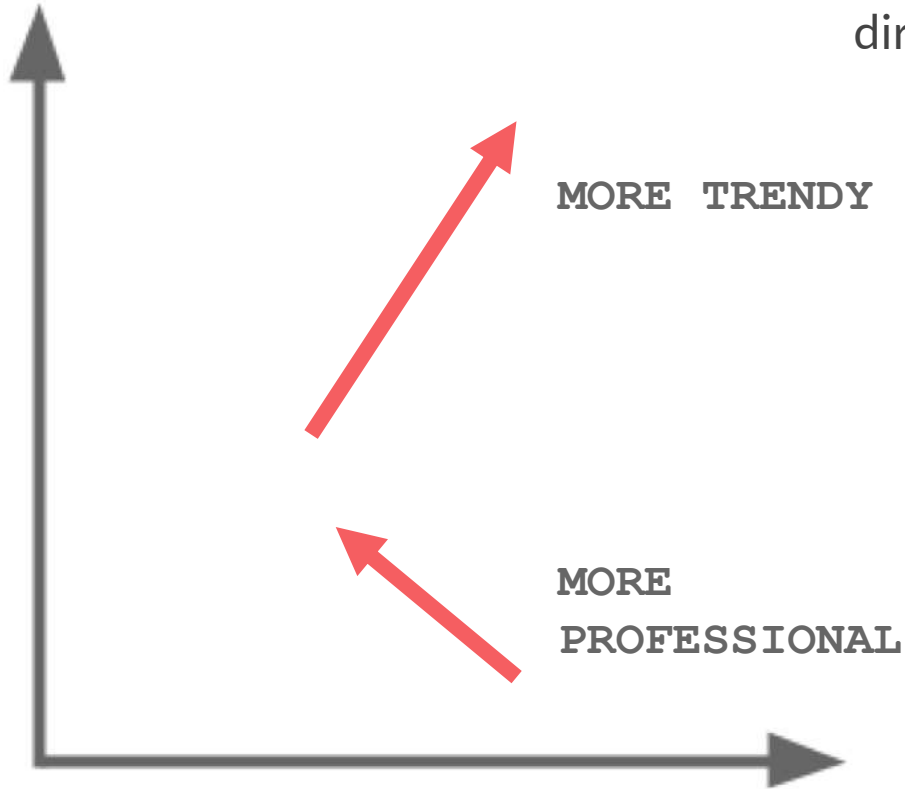
We have hundreds of **directions**  
encoding hundreds of ideas



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?



**Let's try it!**

05 Simple MF Biases is actually word2vec

# 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



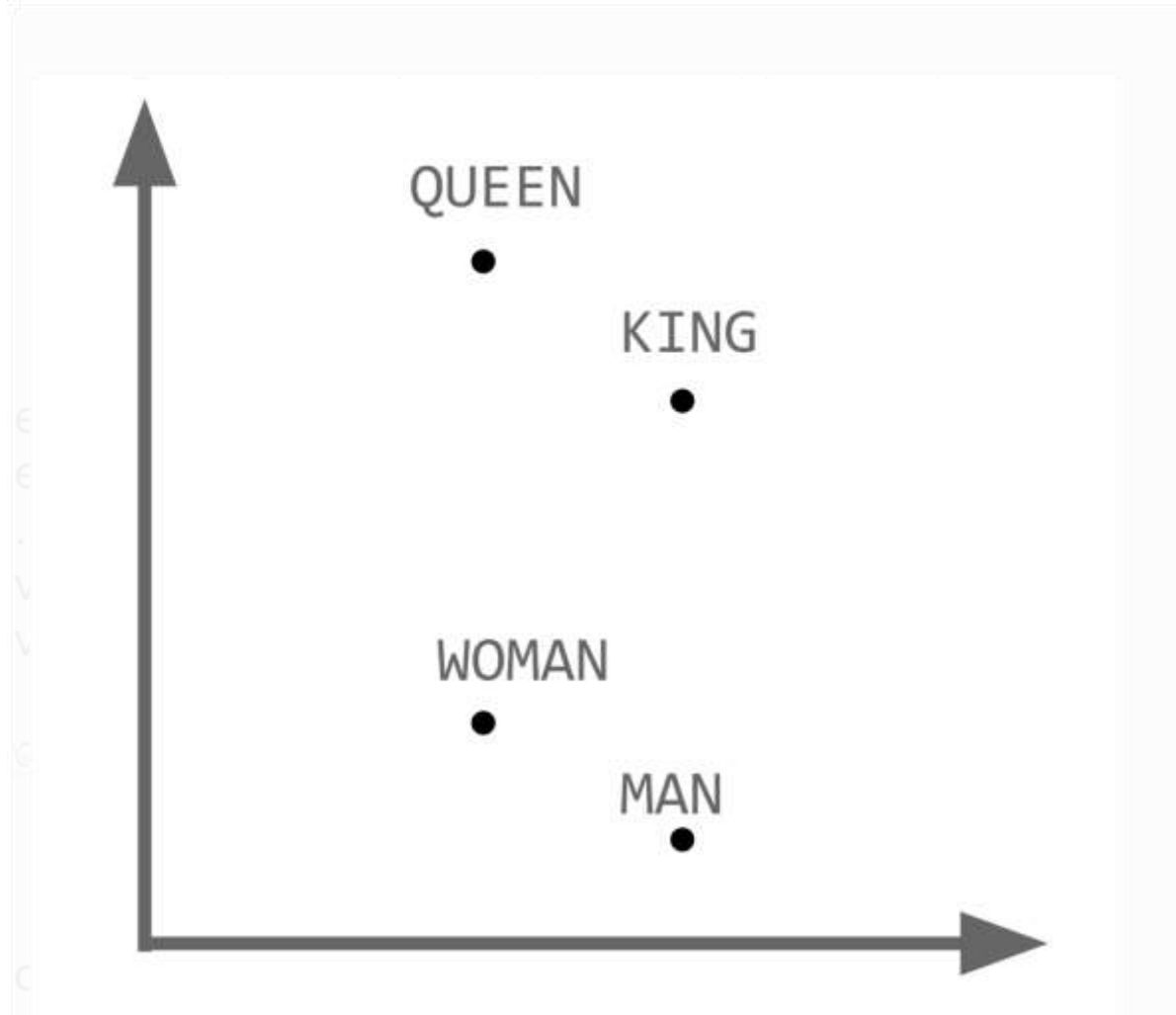
1



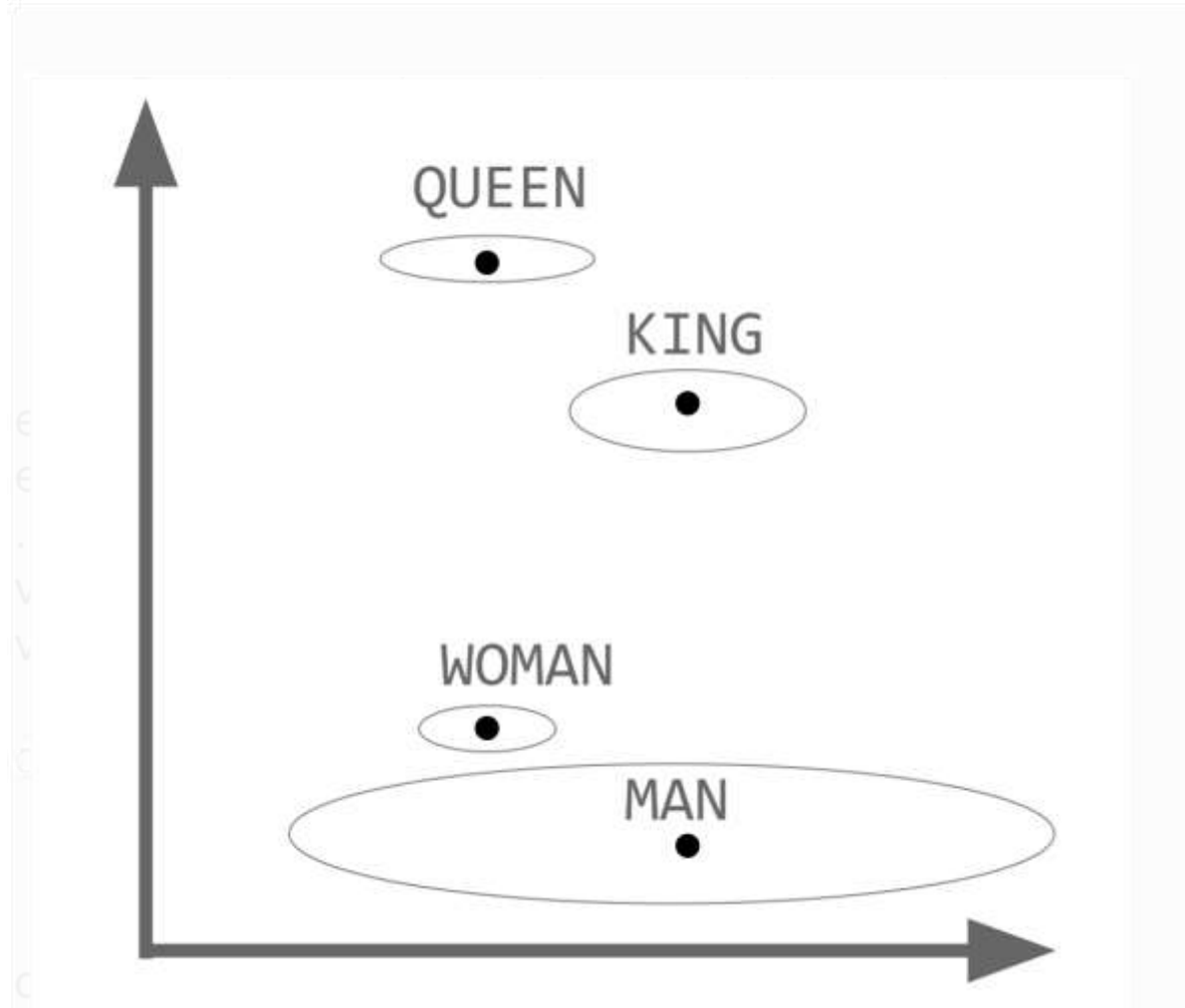
7



22



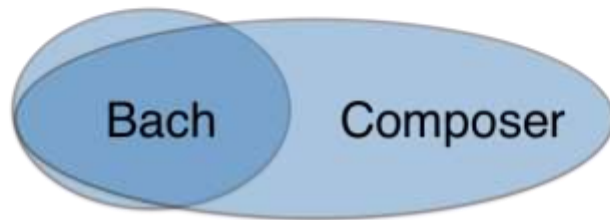




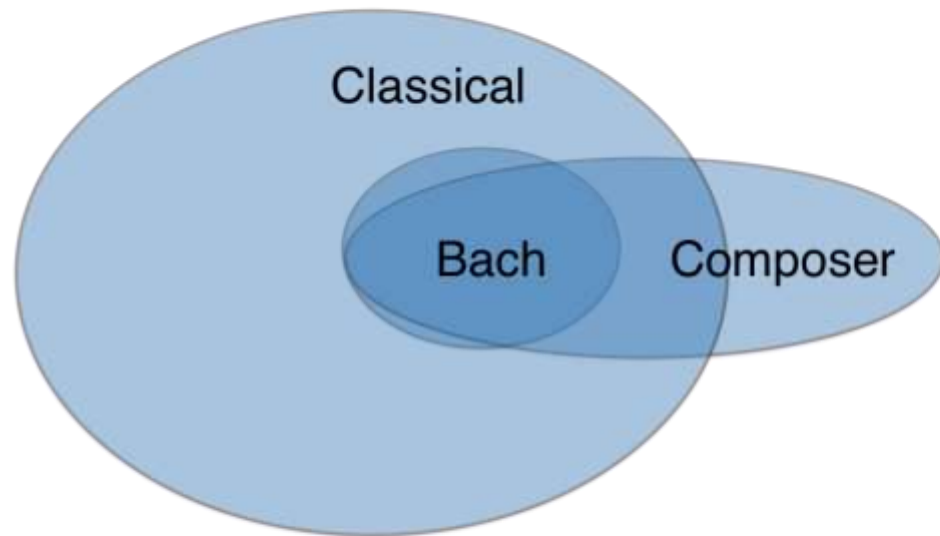
See also:  
'word2gauss'



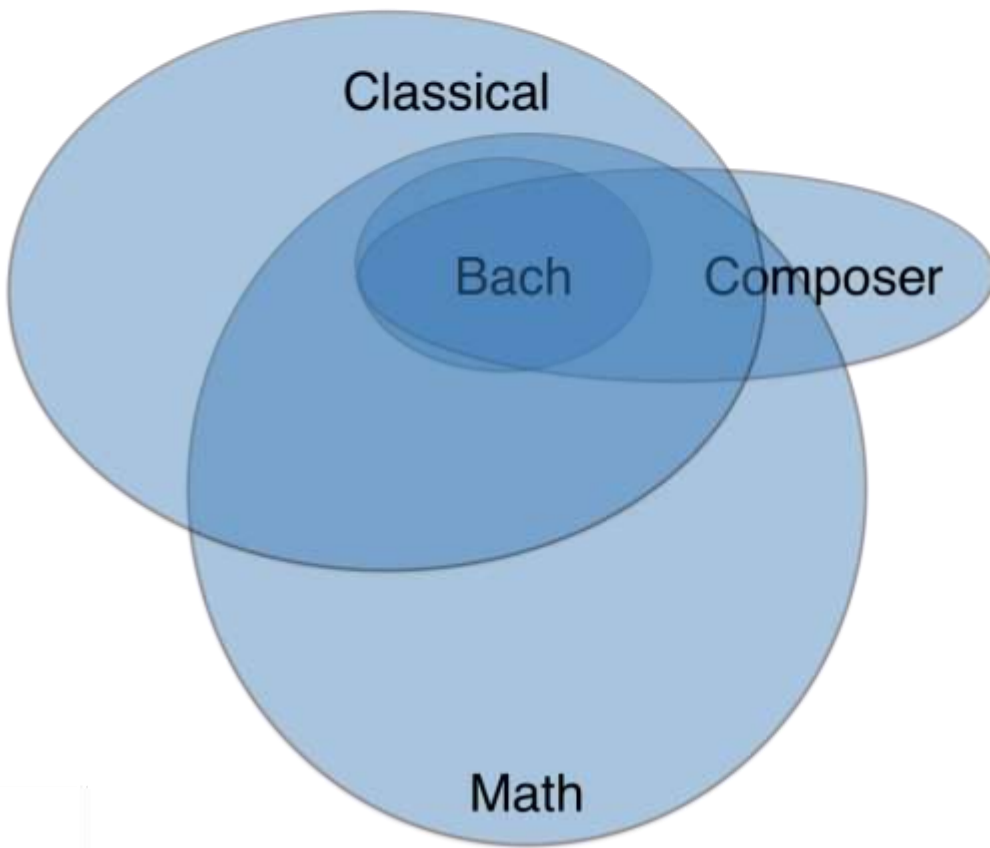
See also:  
'word2gauss'



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)
```

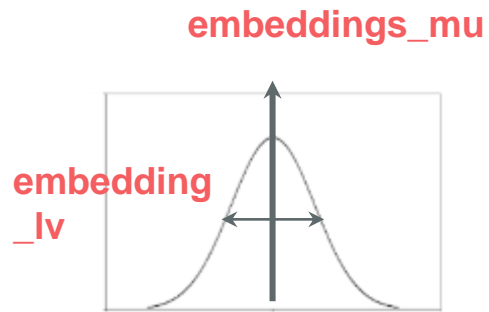
```
embeddings_lv = nn.Embedding(n_users, k)
```

```
...
```

```
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)
```

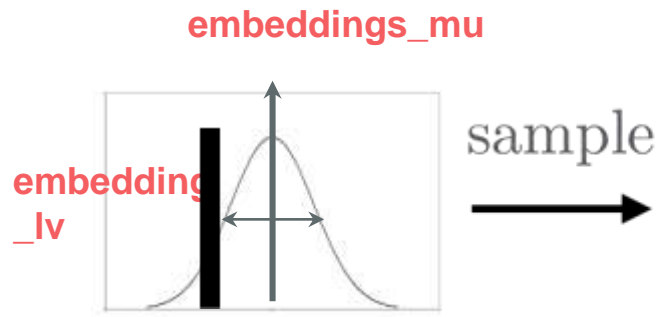
```
...
```

```
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)
```

```
+0.32
```

```
+0.49 ...
```

```
-0.21
```

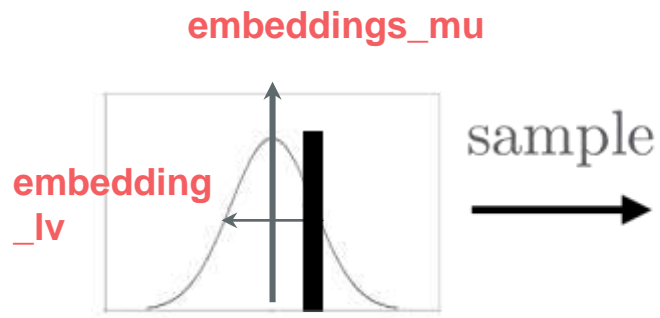
```
+0.03
```

```
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)
```

+0.32

+0.49 ...

-0.21

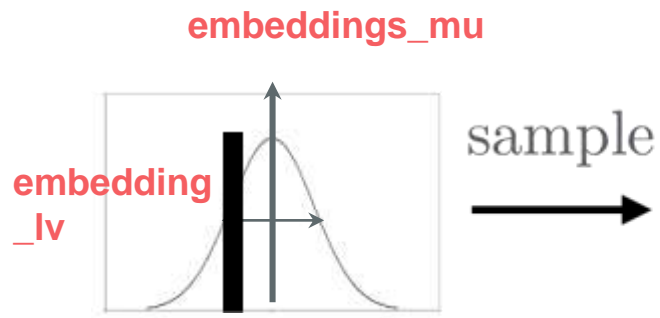
+0.03

```
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)
```

```
+0.32
```

```
+0.49 ...
```

```
-0.21
```

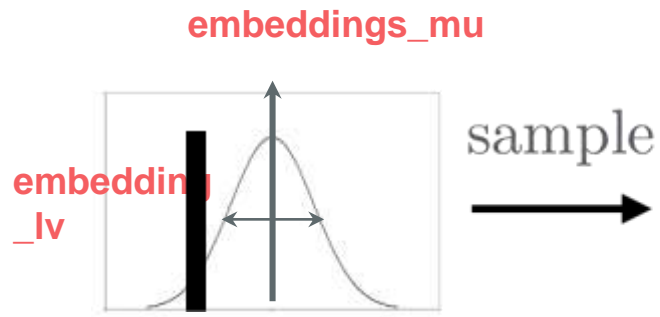
```
+0.03
```

```
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)
```

```
+0.32
```

```
+0.49 ...
```

```
-0.21
```

```
+0.03
```

```
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
```

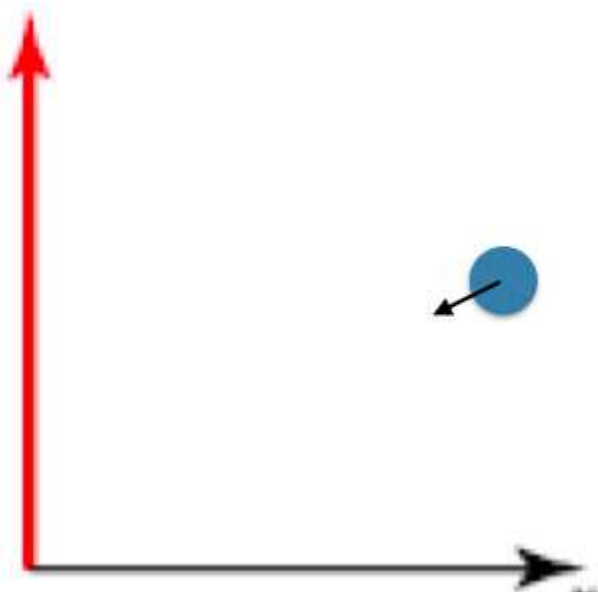
$$R_{ui} = b + \omega_u + \mu_i + p_u \cdot q_i$$

Let's make this variational:

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

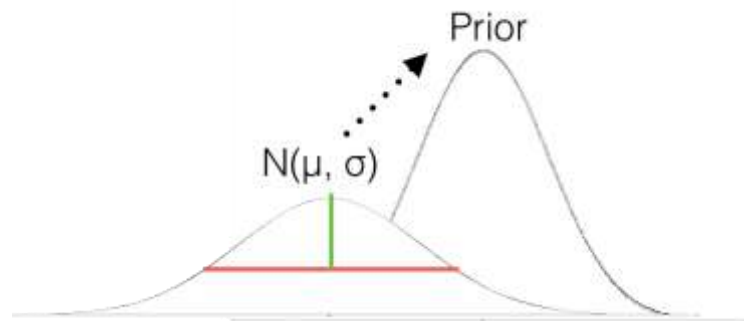
## WITHOUT VARIATIONAL

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



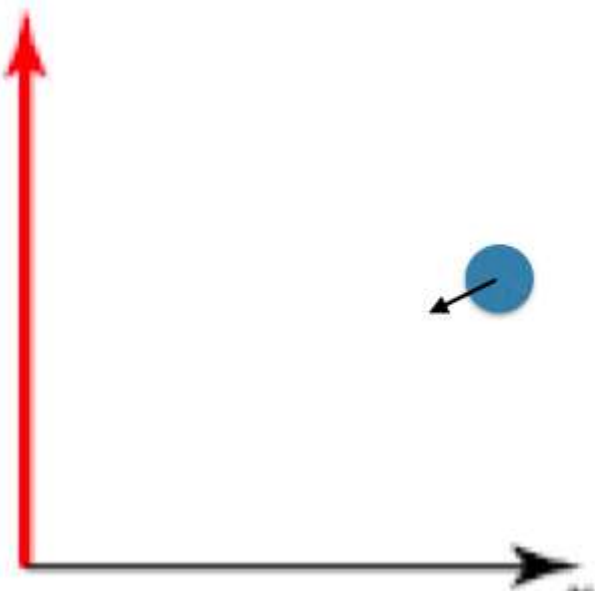
## WITH VARIATIONAL

```
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)
```





At the Frontier

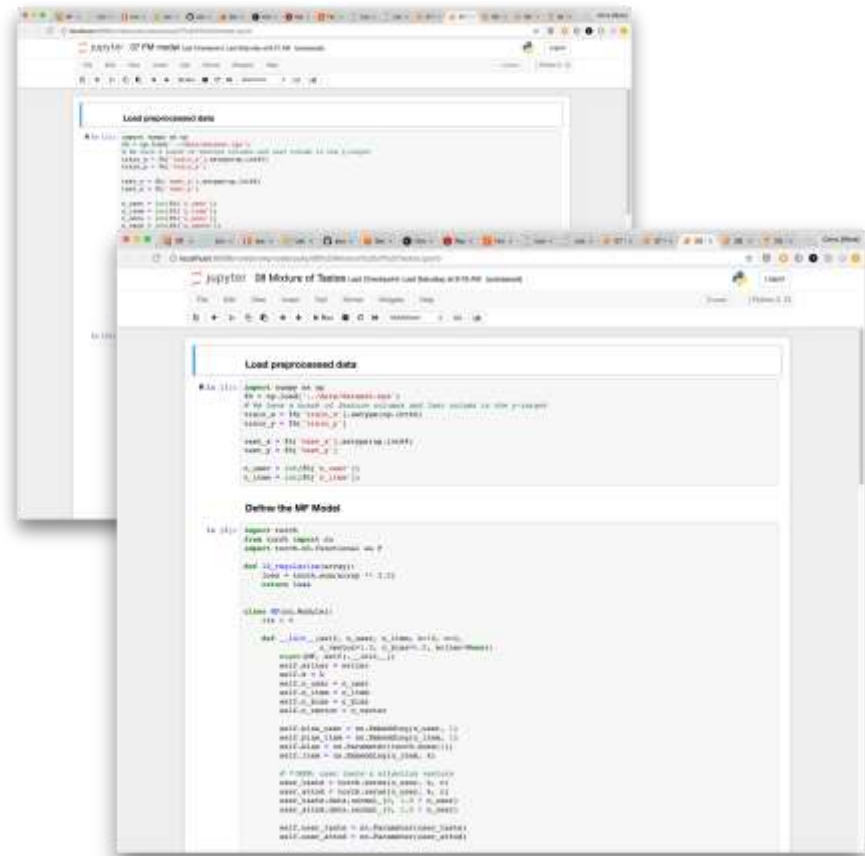
# More Things to Try

## 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  Broaden

[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 Poincaré 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 Poincaré 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:

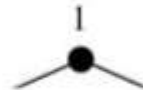
$$V \sim r^d$$



Volume = 9

In Euclidean space, volume grows like:

$$V \sim r^d$$

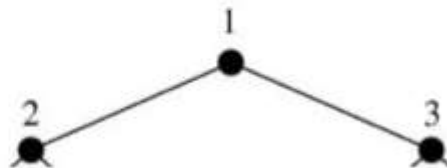


Volume = 1

In binary data structures 'volume' grows like:

$$V \sim 2^r$$



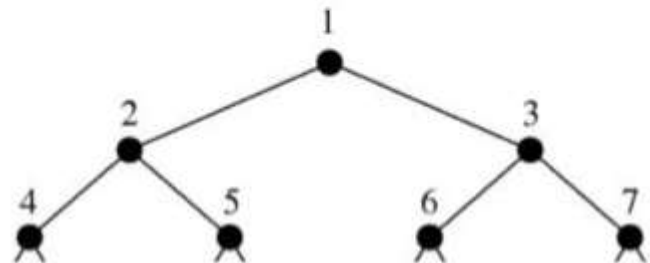


$$\text{Volume} = 2 + 1$$

In binary data structures 'volume' grows like:

$$V \sim 2^r$$

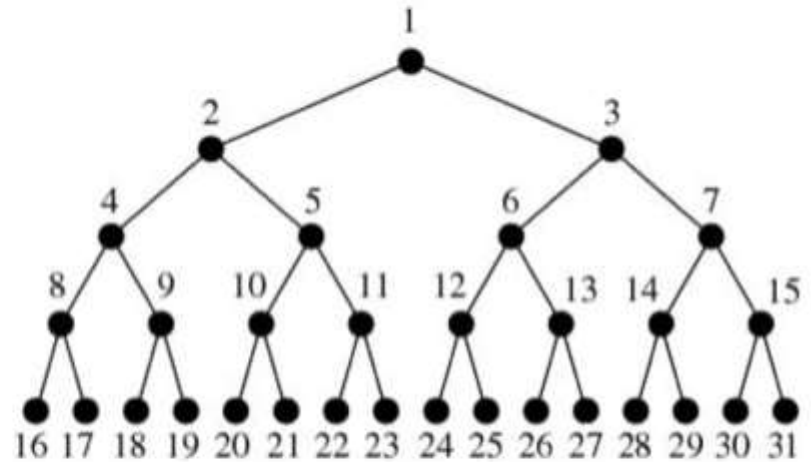
Volume = 4 + 3



In binary data structures 'volume' grows like:

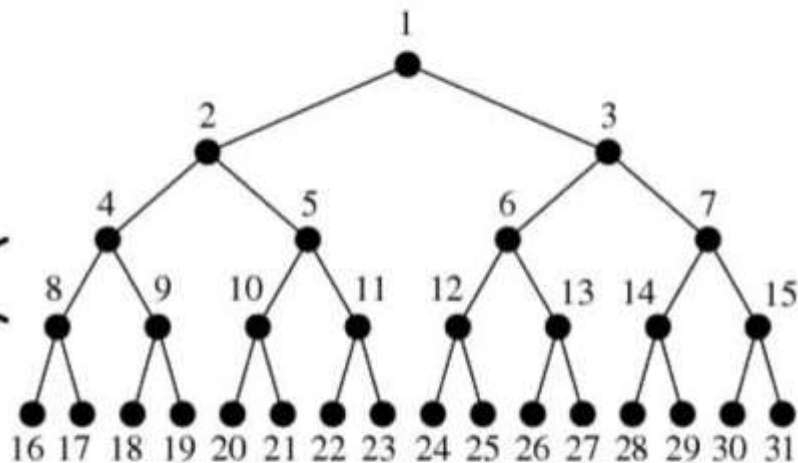
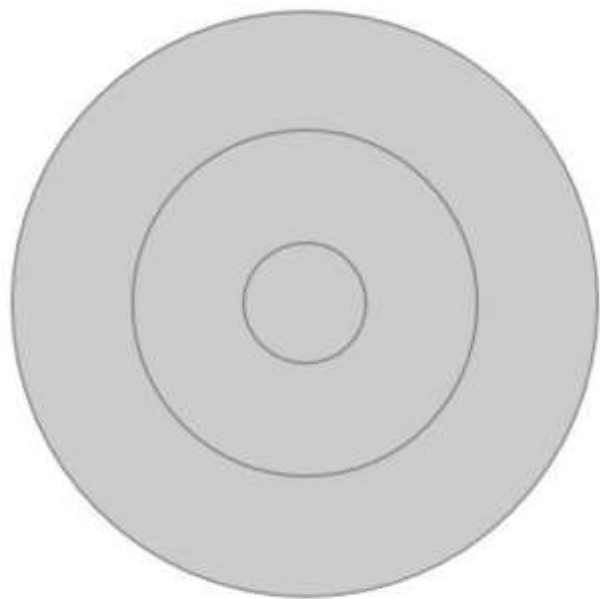
$$V \sim 2^r$$

Volume = 8 + 7



In binary data structures 'volume' grows like:

$$V \sim 2^r$$



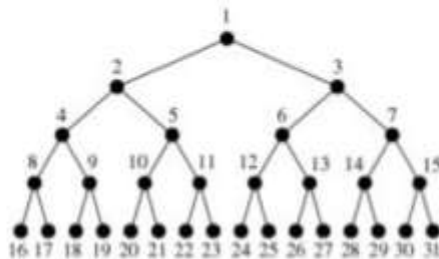
In Euclidean space, volume grows like:

$$V \sim r^d$$

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

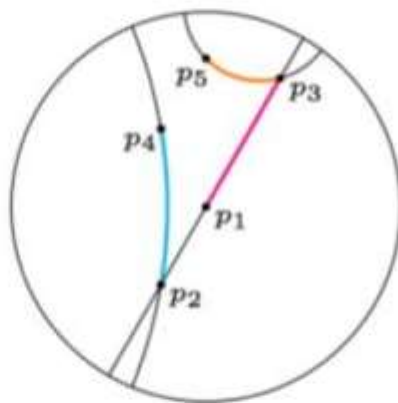


$$V \sim r^d$$

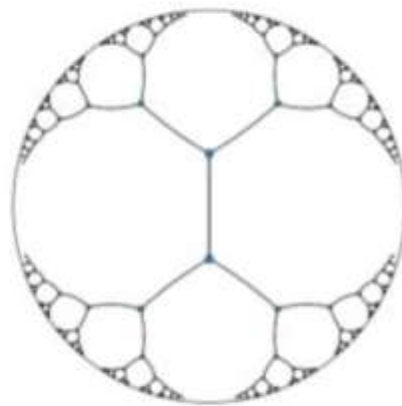
**You need dimensionality that grows exponentially!**

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

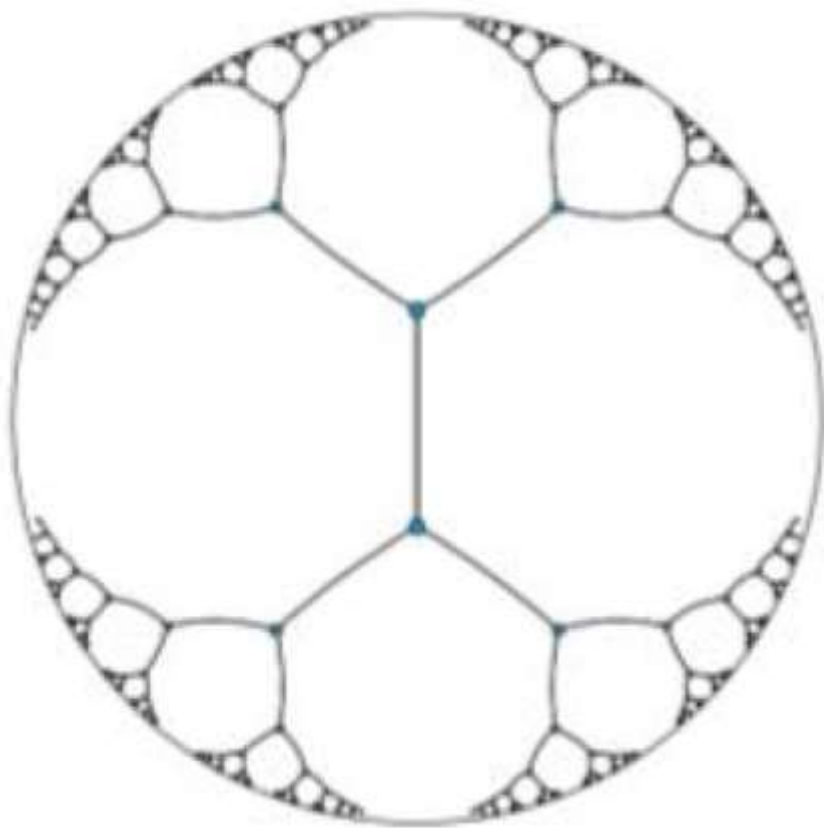
By analogy, hyperbolic  
spaces are continuous  
versions of hierarchies.



(a) Geodesics of the Poincaré disk



(b) Embedding of a tree in  $B^2$



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





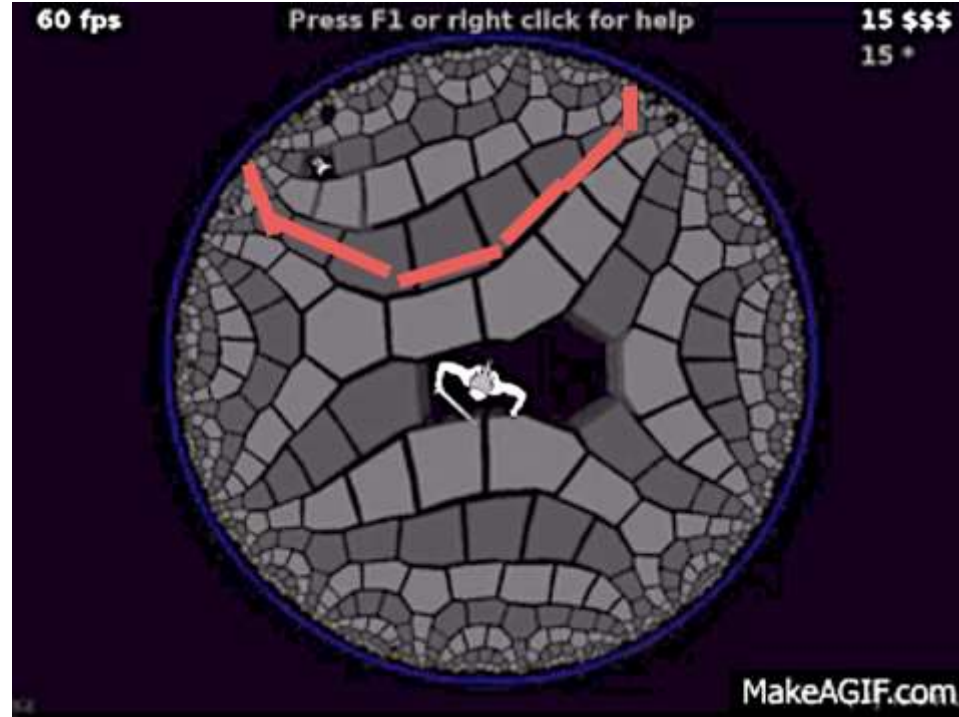
“Hyper Rogue” is a game in hyperbolic space.

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

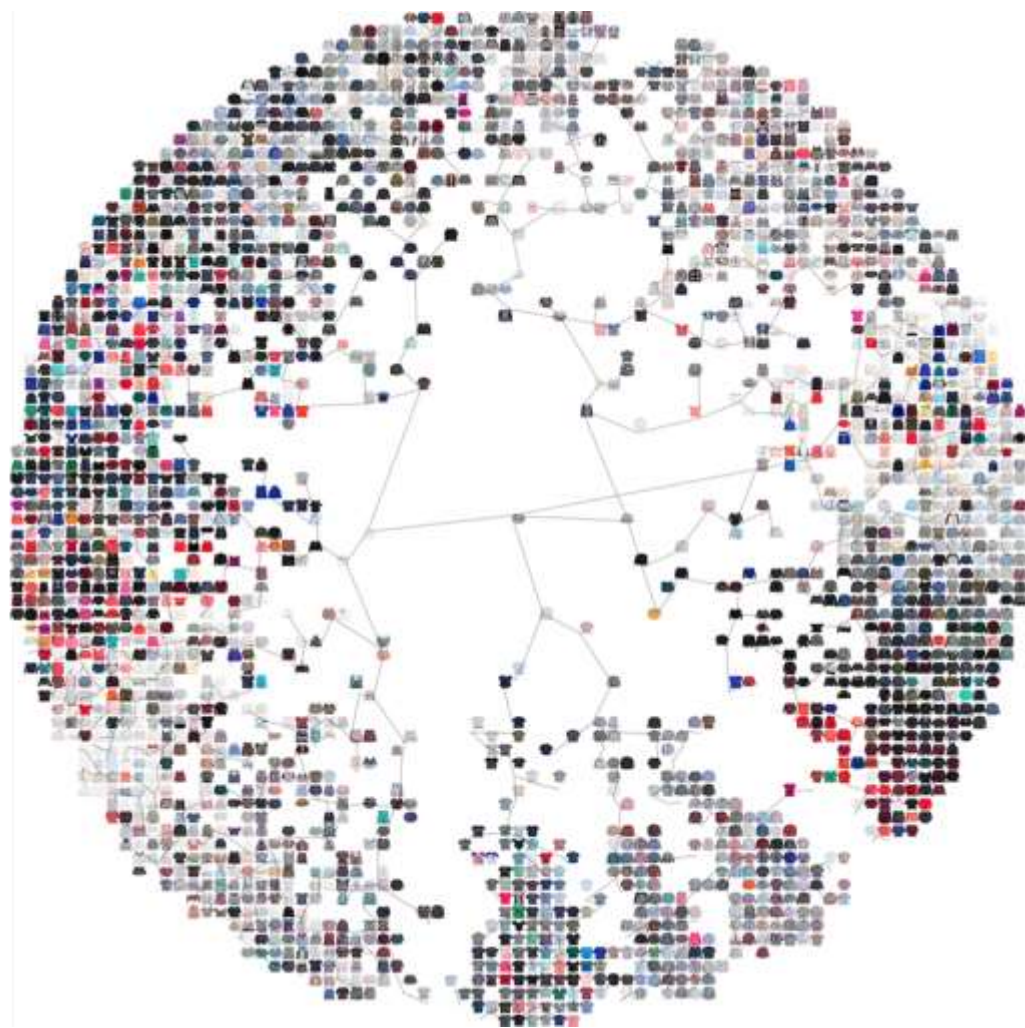


“Hyper Rogue” is a game in hyperbolic space.

- 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