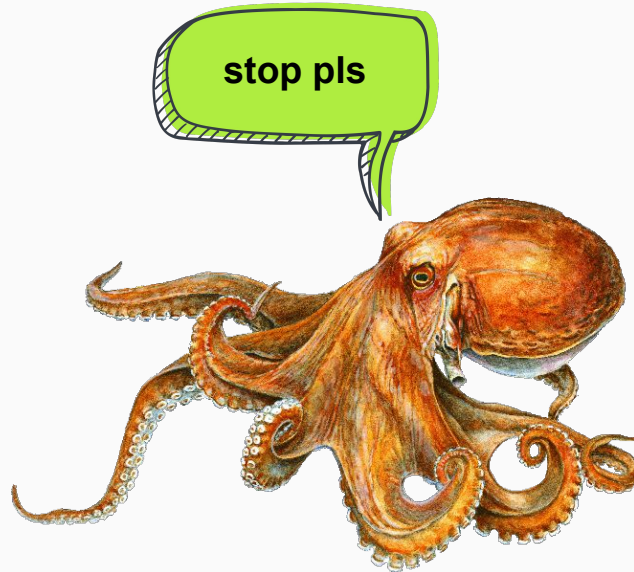


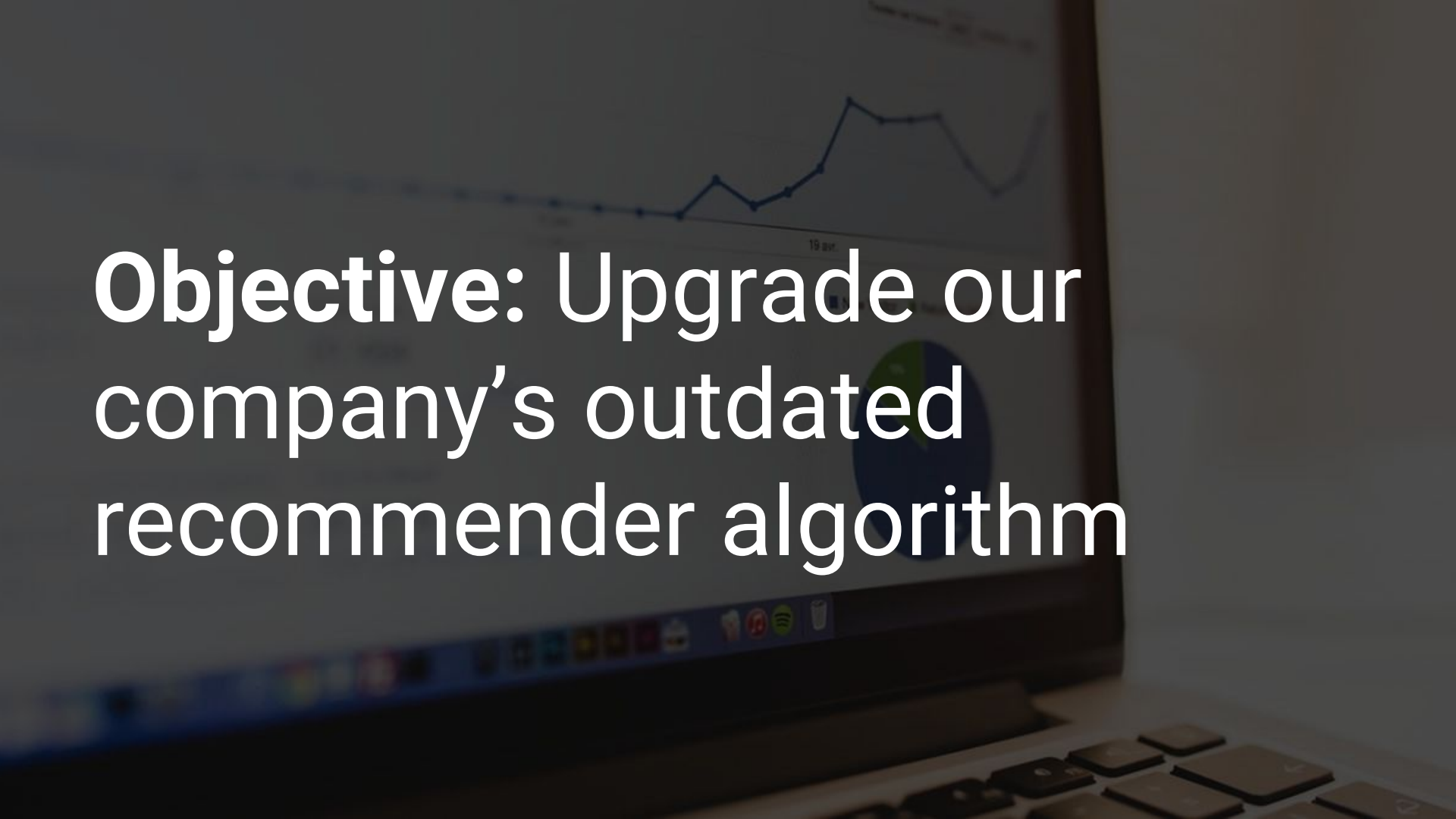
Items-Legit

Recommending recommenders since 1887

How many tickles
does it take to make
an octopus laugh?

Ten-tickles, silly!



The background image shows a laptop screen with a dark overlay. On the screen, there is a line graph with a blue line showing fluctuations over time, and a pie chart with green and blue segments. The text 'Objective: Upgrade our company's outdated recommender algorithm' is written in white, bold, sans-serif font across the center of the screen. The laptop keyboard is visible at the bottom.

Objective: Upgrade our
company's outdated
recommender algorithm

git checkout frank

Aka the team of destiny



Blair Thurman



Tim Kahn



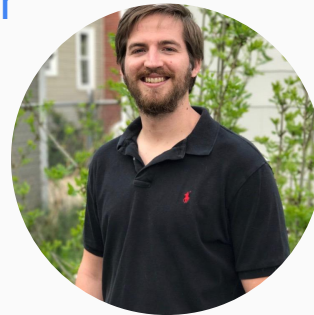
Jennifer Waller



Douglas Schuster



Peter Nygaard

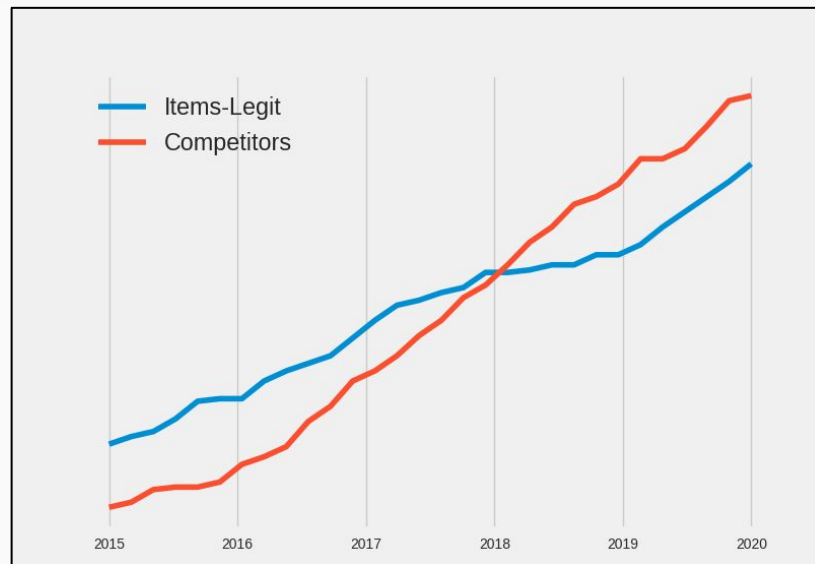


Bob Corboy

The Problem

The method our company uses for the basis of our recommender systems is no longer competitive.

We will continue to lose business if the system is not updated

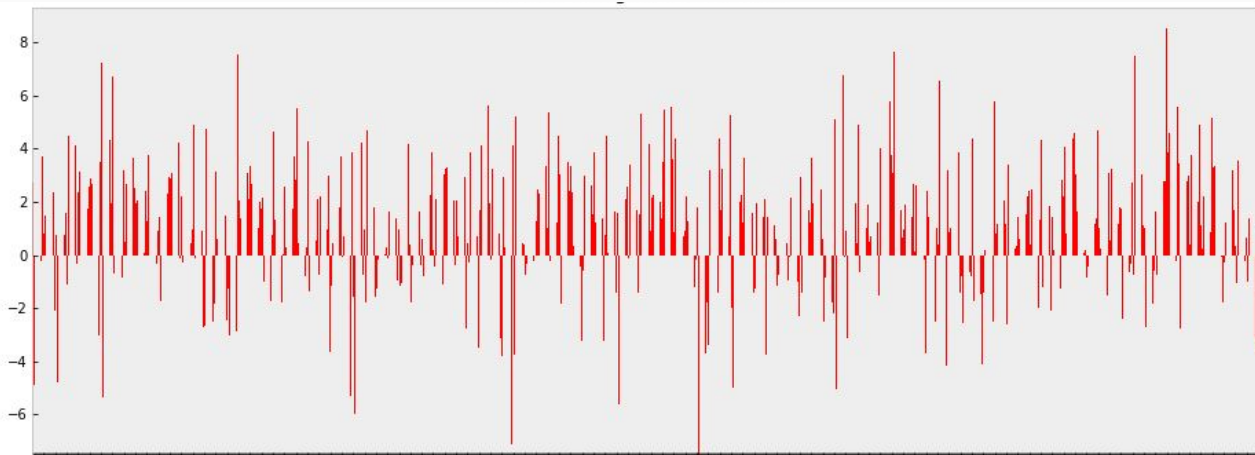


Thinking this through...

Some jokes are more funny than others.



Some users are more easily amused than others.



How to Recommend Things?

	<u>Recommend based on:</u>
Popularity	Items that receive positive ratings or are best-sellers
Collaborative filtering	Ratings from users similar to the user (based on historical ratings) (ex: PMF)
Content-based filtering	Recommend things similar to things liked by the user
Demographic	Demographic characteristics of the user
Knowledge-based	User's needs, how products meet their needs

Latent Features

- **Intuition:** choose descriptors, then find similarity among data (jokes) by looking at their how strongly each joke reflects those descriptors. A human might choose 'cheesy', 'story', 'pun', 'raunchy', etc. and rate how much a joke reflects each category.
- **Recommender algorithms** choose descriptors -- **latent features** -- that best describe the data, given known ratings. Latent features don't always translate well into human vocabulary, but the idea is the same.
- **Probabilistic Matrix Factorization** is a strong algorithm for finding latent features and generating recommendations... in some contexts.

A close-up photograph of a person's hand holding a pen, writing on a document. The background is blurred, showing some bokeh lights. The image is partially covered by a blue overlay on the right side.

Results

We were able to improve on
the old model's RMSE by:

13.7% w/ PMF

14.9% w/ MCMC

60.4% w/ NNLM in R

Baseline



Global Mean
GM

Uniform Random
UR

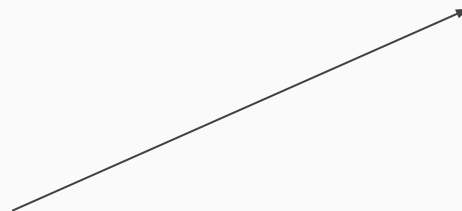
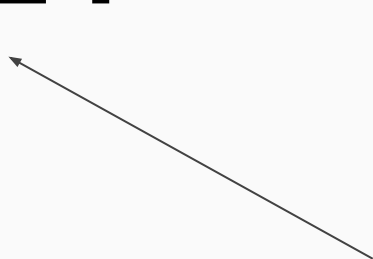
Mean of Means
MoM

5.24

7.79

4.77

RMSE



Probabilistic Matrix Factorization (PMF)

After getting the code to work, we sent it through and got an RMSE of -----> **4.13**



An aerial photograph of the New York City skyline at dusk. The sky is a mix of dark purple, blue, and orange. The city lights are visible, and the water of the harbor is in the background. The Empire State Building is prominent in the center-left.

The solution:
PMF in PYMC3
... and NNLM in R?

PCMC3

Once we got the MCMC
portion of the code to work,
we got an even better
improvement of 16.9%!!!!

4.8

R

nnlm package

“NMF with Stochastic Gradient Descent is often the best solution for recommenders because SGD can elegantly avoid NaNs when fitting

UVD + SGD makes a lot of sense for recommender systems.

In fact, NMF + SGD is ‘best in class’ option for many recommender domains:

- No need to impute missing values.
- Use regularization to avoid overfitting.
- Optionally include biases terms to communicate prior knowledge.
- Can handle time-dynamics (e.g. change in user preference over time).
- Used by the winning entry in the Netflix challenge.)”

[1]

```
ratings = read.csv('dsi/github_repos/alt-recommender-case-study/
                  data/movies/ratings_new_dense.csv')
```

```
x_movies <- t(t(seq(1, 671, by=1)))
```

```
y_movies <- data.matrix(ratings, rownames.force = NA)
```

```
nnlm(x_movies, y_movies, alpha = rep(0,3), method = c('scd', 'lee'),
     loss = c('mse', 'mkl'), init = NULL, mask = NULL,
     check.x = TRUE, max.iter = 10000L, rel.tol = 1e-12,
     n.threads = 1L)
```

```
$n.iteration
[1] 18134
```

```
$error
```

```
MSE
3.654381
```

```
MKL target.error
NA      1.827190
```

```
$options
```

```
$options$method
[1] "scd"
```

```
$options$loss
[1] "mse"
```

```
jester_dataset = read.csv('dsi/github_repos/alt-recommender-case-study/
                          data/jokes/jester-dataset-v1-dense-first-1000.csv')
```

```
x_jokes <- t(t(seq(1, 100, by=1)))
```

```
y_jokes <- t(data.matrix(jester_dataset, rownames.force = NA))
```

```
nnlm(x_jokes, y_jokes, alpha = rep(0,3), method = c('scd', 'lee'),
     loss = c('mse', 'mkl'), init = NULL, mask = NULL,
     check.x = TRUE, max.iter = 10000L, rel.tol = 1e-12,
     n.threads = 1L)
```

```
$n.iteration
[1] 2000
```

```
$error
```

```
MSE
23.86182
```

```
MKL target.error
NaN      11.93091
```

```
$options
```

```
$options$method
[1] "scd"
```

```
$options$loss
[1] "mse"
```


Pro & of PMF s and PYMC3



- **PRO:** Ability to create a well-tuned model with accurate recommendations; finding a MAP estimate of model parameters is reasonably efficient, even in very large data sets.
- **CON:** Prone to overfitting unless regularization params carefully tuned
- **CON:** High computational cost out-of-box
 - Can be overcome with methodology (ADVI) and use of GPU.
- **CON:** Human learning curve / implementation difficulties
 - Probabilistic programming is a very different approach for most
 - Difficulty handling sparse data with out-of-box PMF method

Implementation

Steps and timeframe for
materialization of upgraded product

Prototype - 2 months

Data scientist team creates the framework for the algorithm

Production - 4 months

Software developers develop the solution

Deploy to customers

Recommenders are available for client use

Questions?

PMF vs. Bayesian PMF

