Reddicommend

Content recommendations for Reddit

Motivation

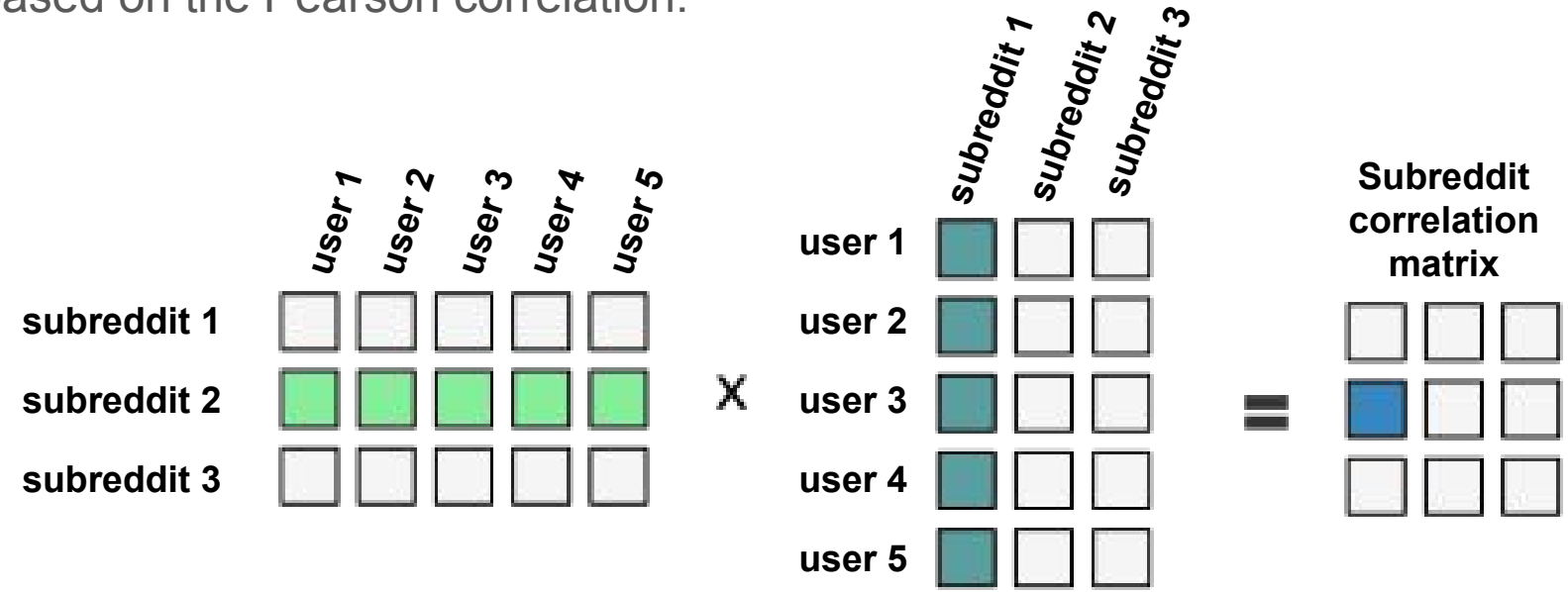
- Reddit is the fourth-most visited site in the US (8th globally)
- The ability of users to find content relevant to their interests is important to its utility as a content-aggregating site
- *Reddicommend* is an engine for personalized subreddit recommendations

Pipeline


S3
Flask
web development,
one drop at a time
APACHE
Spark™
MySQL®

Implementation

Collaborative filtering using similarity measure based on the Pearson correlation:



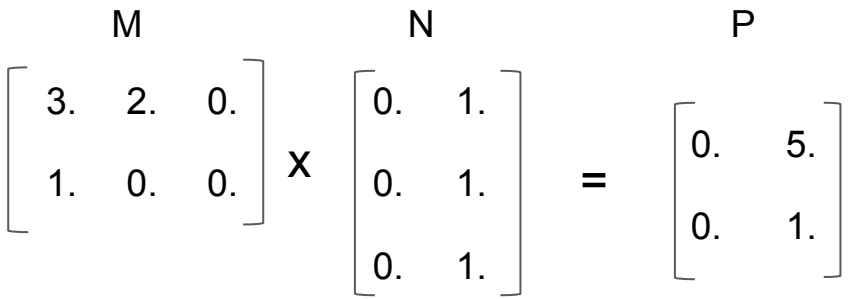
Scalable matrix multiplication in Spark

- Managing time complexity
  - Dense-matrix multiplication is $O(n^3)$ (for square matrices)
    - > 1 million users per batch
    - > 100,000 subreddits

Scalable matrix multiplication in Spark

$$MN = P$$

$$p_{ik} = \sum_j m_{ij} n_{jk}$$



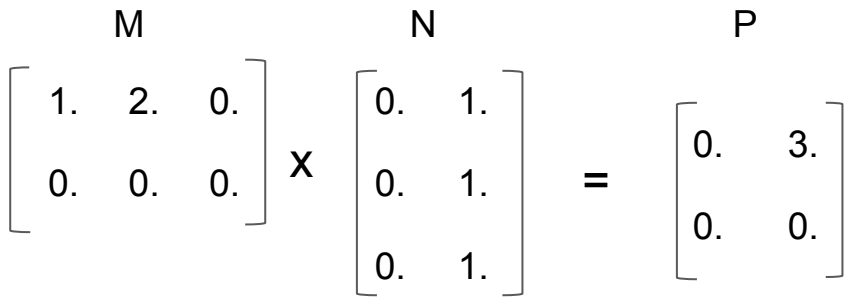
M: [MatrixEntry(0, 0, 3), MatrixEntry(0, 1, 2), MatrixEntry(1, 0, 1)]
N: [MatrixEntry(0, 1, 1), MatrixEntry(1, 1, 1), MatrixEntry(2, 1, 1)]

```
val M = leftMatrix.entries.map({ case MatrixEntry(i, j, v) => (j, (i, v)) })
val N = rightMatrix.entries.map({ case MatrixEntry(j, k, w) => (j, (k, w)) })
val product_entries = M
  .join(N)
  .map({ case (_, ((i, v), (k, w))) => ((i, k), (v * w)) })
  .reduceByKey(_ + _)
  .filter({case ((i, k), p_ik) => p_ik > zero_threshold})
  .map({ case ((i, k), p_ik) => MatrixEntry(i, k, p_ik) })
```

Scalable matrix multiplication in Spark

$$MN = P$$

$$p_{ik} = \sum_j m_{ij} n_{jk}$$



M: [MatrixEntry(0, 0, 1), MatrixEntry(0, 1, 2)] → [(0, (0, 1)), (1, (0, 2)]

N: [MatrixEntry(0, 1, 1), MatrixEntry(1, 1, 1), MatrixEntry(2, 1, 1)] → [(0, (1, 1)), (1, (1, 1)), (2, (1, 1))]

```
val M = leftMatrix.entries.map({ case MatrixEntry(i, j, v) => (j, (i, v)) })
val N = rightMatrix.entries.map({ case MatrixEntry(j, k, w) => (j, (k, w)) })
val product_entries = M
  .join(N)
  .map({ case (_, ((i, v), (k, w))) => ((i, k), (v * w)) })
  .reduceByKey(_ + _)
  .filter({case ((i, k), p_ik) => p_ik > zero_threshold})
  .map({ case ((i, k), p_ik) => MatrixEntry(i, k, p_ik) })
```

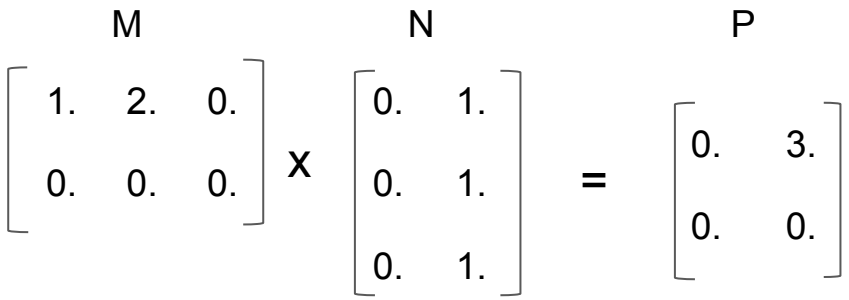
Scalable matrix multiplication in Spark

$$MN = P$$

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

[(0, (0, 1)), (1, (0, 2)]
[(0, (1, 1)), (1, (1, 1)), (2, (1, 1))]

→ [(0, ((0, 1), (1, 1))), (1, ((0, 2), (1, 1)))]




```
val M = leftMatrix.entries.map({ case MatrixEntry(i, j, v) => (j, (i, v)) })
val N = rightMatrix.entries.map({ case MatrixEntry(j, k, w) => (j, (k, w)) })
val product_entries = M
  .join(N)
  .map({ case (_, ((i, v), (k, w))) => ((i, k), (v * w)) })
  .reduceByKey(_ + _)
  .filter({case ((i, k), p_ik) => p_ik > zero_threshold})
  .map({ case ((i, k), p_ik) => MatrixEntry(i, k, p_ik) })
```

Scalable matrix multiplication in Spark

$$MN = P$$

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

M

$$\begin{bmatrix} 1. & 2. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

x

N

$$\begin{bmatrix} 0. & 1. \\ 0. & 1. \\ 0. & 1. \end{bmatrix}$$

=

P

$$\begin{bmatrix} 0. & 3. \\ 0. & 0. \end{bmatrix}$$

[(0, ((0, 1), (1, 1))), (1, ((0, 2), (1, 1)))] → [((0, 1), 1), ((0, 1), 2)]

```
val M = leftMatrix.entries.map({ case MatrixEntry(i, j, v) => (j, (i, v)) })
val N = rightMatrix.entries.map({ case MatrixEntry(j, k, w) => (j, (k, w)) })
val product_entries = M
  .join(N)
  .map({ case (_, ((i, v), (k, w))) => ((i, k), (v * w)) })
  .reduceByKey(_ + _)
  .filter({case ((i, k), p_ik) => p_ik > zero_threshold})
  .map({ case ((i, k), p_ik) => MatrixEntry(i, k, p_ik) })
```

Scalable matrix multiplication in Spark

$$MN = P$$

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

M

$$\begin{bmatrix} 1. & 2. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

x

N

$$\begin{bmatrix} 0. & 1. \\ 0. & 1. \\ 0. & 1. \end{bmatrix}$$

=

P

$$\begin{bmatrix} 0. & 3. \\ 0. & 0. \end{bmatrix}$$

[(0, ((0, 1), (1, 1))), (1, ((0, 2), (1, 1)))] → [((0, 1), 1), ((0, 1), 2)] → [((0, 1), 3)]

```
val M = leftMatrix.entries.map({ case MatrixEntry(i, j, v) => (j, (i, v)) })
val N = rightMatrix.entries.map({ case MatrixEntry(j, k, w) => (j, (k, w)) })
val product_entries = M
  .join(N)
  .map({ case (_, ((i, v), (k, w))) => ((i, k), (v * w)) })
  .reduceByKey(_ + _)
  .filter({case ((i, k), p_ik) => p_ik > zero_threshold})
  .map({ case ((i, k), p_ik) => MatrixEntry(i, k, p_ik) })
```

Scalable matrix multiplication in Spark

$$MN = P$$

$$p_{ik} = \sum_j m_{ij} n_{jk}$$

M

$$\begin{bmatrix} 1. & 2. & 0. \\ 0. & 0. & 0. \end{bmatrix}$$

x

N

$$\begin{bmatrix} 0. & 1. \\ 0. & 1. \\ 0. & 1. \end{bmatrix}$$

=

P

$$\begin{bmatrix} 0. & 3. \\ 0. & 0. \end{bmatrix}$$

[(0, ((0, 1), (1, 1))), (1, ((0, 2), (1, 1)))] → [((0, 1), 1), ((0, 1), 2)] → [((0, 1), 3)]

→ [MatrixEntry(0, 1, 3)]

```
val M = leftMatrix.entries.map({ case MatrixEntry(i, j, v) => (j, (i, v)) })
val N = rightMatrix.entries.map({ case MatrixEntry(j, k, w) => (j, (k, w)) })
val product_entries = M
  .join(N)
  .map({ case (_, ((i, v), (k, w))) => ((i, k), (v * w)) })
  .reduceByKey(_ + _)
  .filter({case ((i, k), p_ik) => p_ik > zero_threshold})
  .map({ case ((i, k), p_ik) => MatrixEntry(i, k, p_ik) })
```

Demo

reddicommend.ddns.net

Oliver Hoidn

- B.S. in Physics, Harvey Mudd College
- PhD in Physics, University of Washington

Past and current interests:

- Hiking
- Violin
- Giant pumpkin cultivation
- Lisp and functional programming

HARVEY MUDD COLLEGE

(lisp)