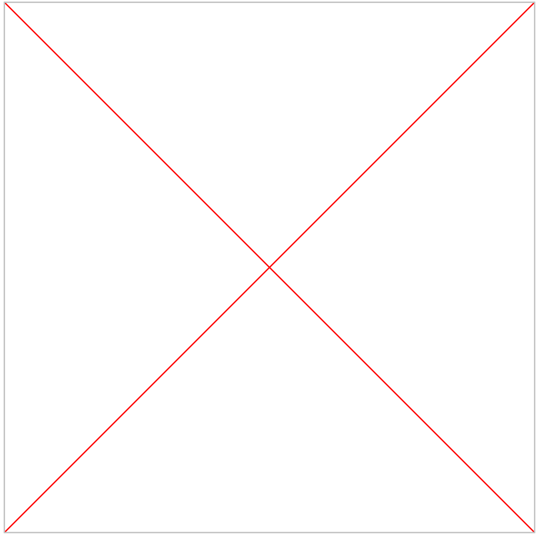
# Recommendation Engines Explained

Rui Vieira
Sophie Watson

rui@redhat.com
sophie@redhat.com

# Jupyter notebook



https://github.com/ruivieira/workshop-recommendation-engines

NETFLIX

goodreads

Expedia®

TESCO

hulu

# Overview

- Collaborative Filtering
  - Alternating least squares
    - Parameter tuning
    - Metrics
- Implicit recommendations
- Post Processing
- Streaming Data

# Python + Apache Spark

Fast

General

Easy

SQL

Text Processing

Machine Learning

APACHE Spark™

# Notebooks

jupyter workshop (unsaved changes)

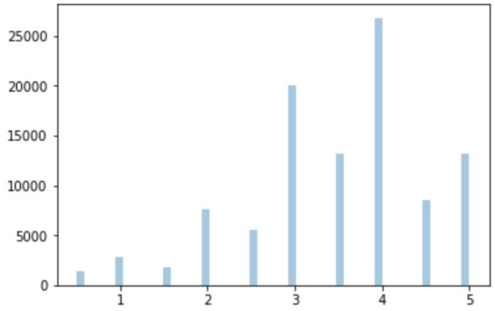
File Edit View Insert Cell Kernel Widgets Help    Not Trusted    Python 3

Run    Markdown

In [4]:
```
import seaborn as sns

r = ratings.select('rating').collect()
sns.distplot(r, kde=False)
```

Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x124eeb160>




We can see that most of the ratings are $4$ starts, followed by $3$ and $3.5$ stars. Users generally do not give very low ratings (e.g. $0.5$, $1$).

Let's try to see now the distribution of the number of ratings per user. That is, so user give many ratings on average?

# Data

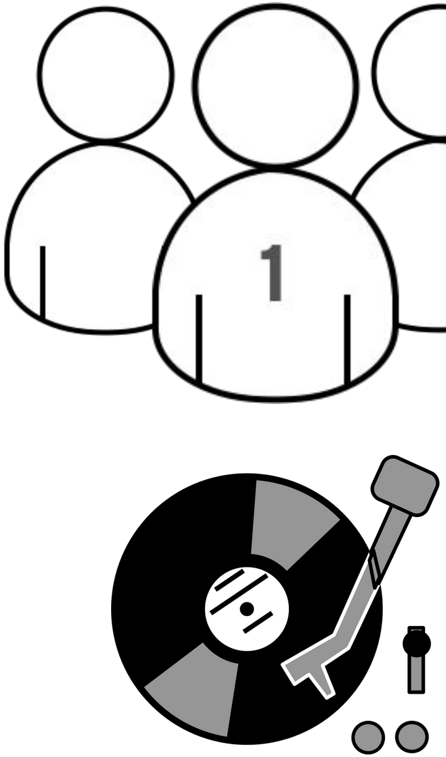
- **MovieLens** [1]
- Widely used in recommendation engine research
- Variants
  - Small - 100,000 ratings / 9,000 movies / 700 users
  - Full - 26 million ratings / 45,000 movies / 270,000 users
- **CSV data**
  - Ratings
    - `(userId, movieId, rating, timestamp)`
    - `(100, 200, 3.5, 2010-12-10 12:00:00)`

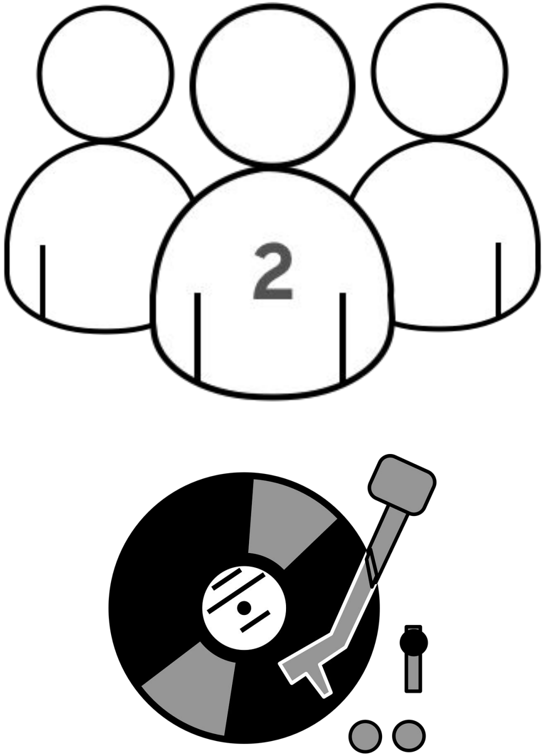
[1] - https://grouplens.org/datasets/movielens/

# Loading in the Data

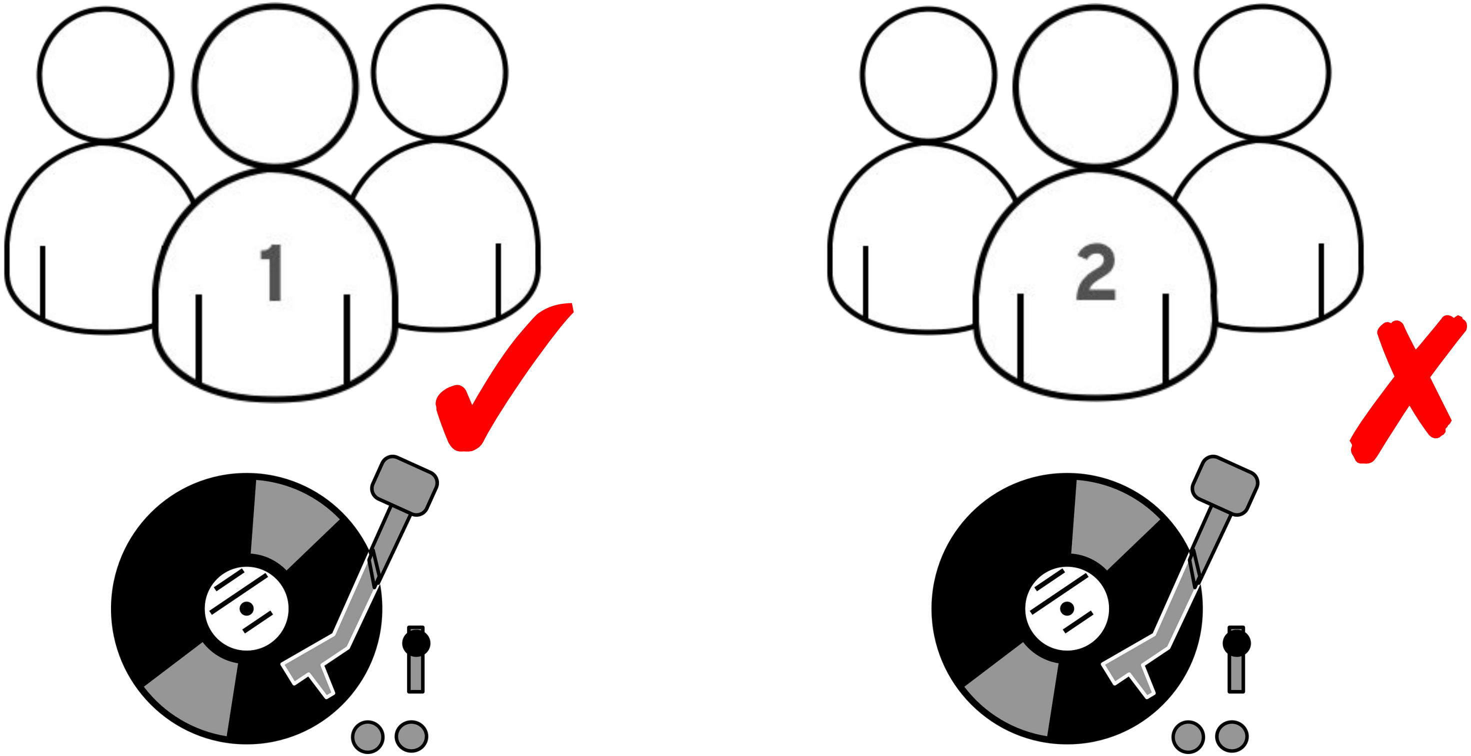

C

# Collaborative Filtering



# Collaborative Filtering

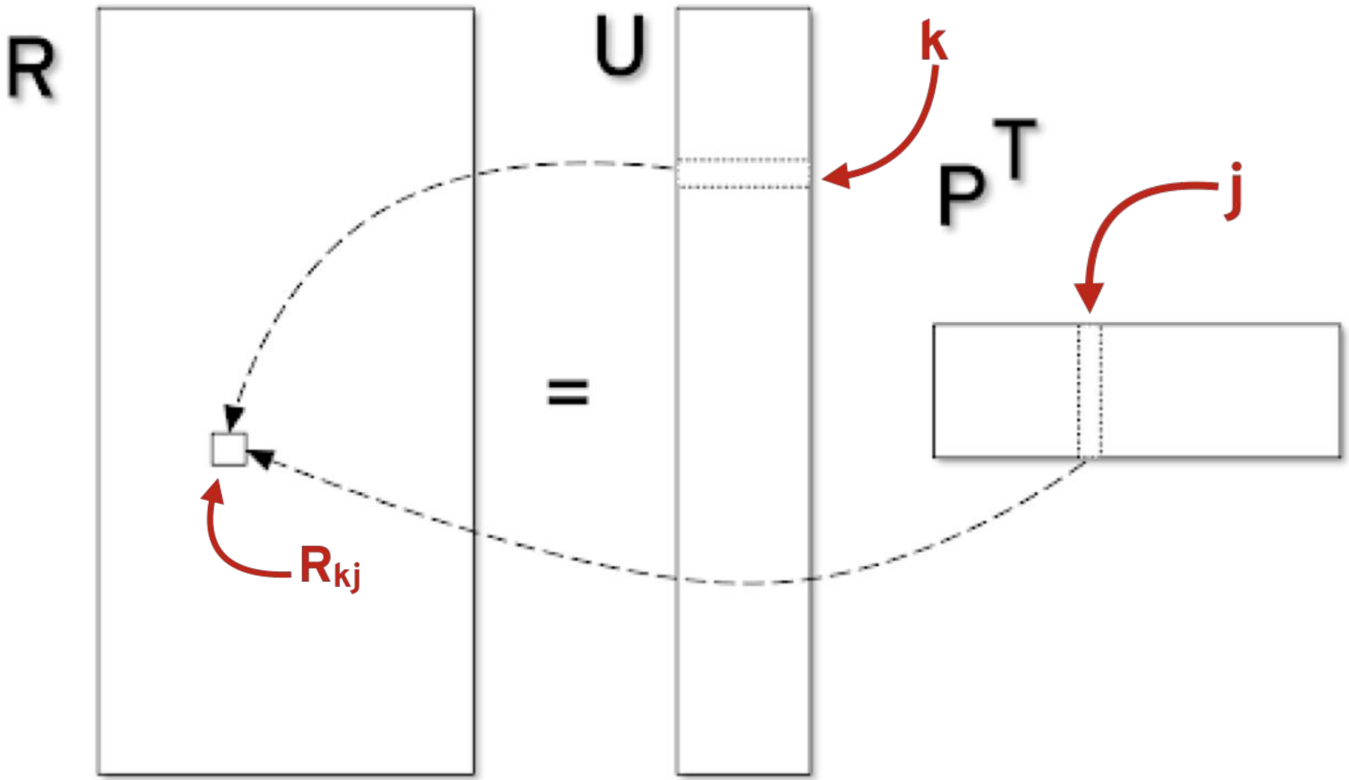


# Collaborative Filtering

- (user, product) → rating
- users with similar “tastes” → good bet
- user and product agnostic

# Alternating Least Squares?

$$
\begin{array}{c c}
 & \begin{array}{ccccc} \text{user } 1 & \text{user } 2 & \text{user } 3 & \cdots & \text{user } N \end{array} \\
R = & \left[\begin{array}{ccccc} 1 & 4.5 & ? & \cdots & 3 \\ ? & 3 & 3 & \cdots & 4 \\ 5 & 3 & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \cdots & ? \end{array}\right] \begin{array}{l} \text{product } 1 \\ \text{product } 2 \\ \text{product } 3 \\ \vdots \\ \text{product } M \end{array}
\end{array}
$$


R
U
$P^T$
=
k
j
$R_{kj}$

# Alternating Least Squares?

$$
R = \begin{array}{c} \\ \\ \\ \\ \\ \\ \end{array}
\begin{array}{cccccc}
\text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user N} & \\
\end{array}
$$

$$
R = \begin{array}{c|c}
 & \begin{array}{ccccc} \text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user N} \end{array} \\
 & \left[\begin{array}{ccccc}
1 & 4.5 & \color{red}{3.8} & \cdots & 3 \\
\color{red}{3.2} & 3 & 3 & \cdots & 4 \\
5 & 3 & \color{red}{3.4} & \cdots & \color{red}{3.1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
2 & 4 & 1 & \cdots & \color{red}{2.7}
\end{array}\right]
\begin{array}{c} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product M} \end{array}
\end{array}
$$

# ALS in Spark

```
#import the class
from pyspark.ml.recommendation import ALS

#initialise the model
simple_als = ALS(maxIter=5, regParam=0.01, rank=3, coldStartStrategy='drop')

#train the model
model = simple_als.fit(sets['training'])

#make predictions
prediction = model.transform(sets['test'])
```

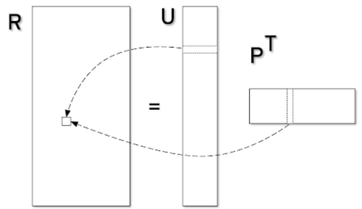
# Mean Squared Error

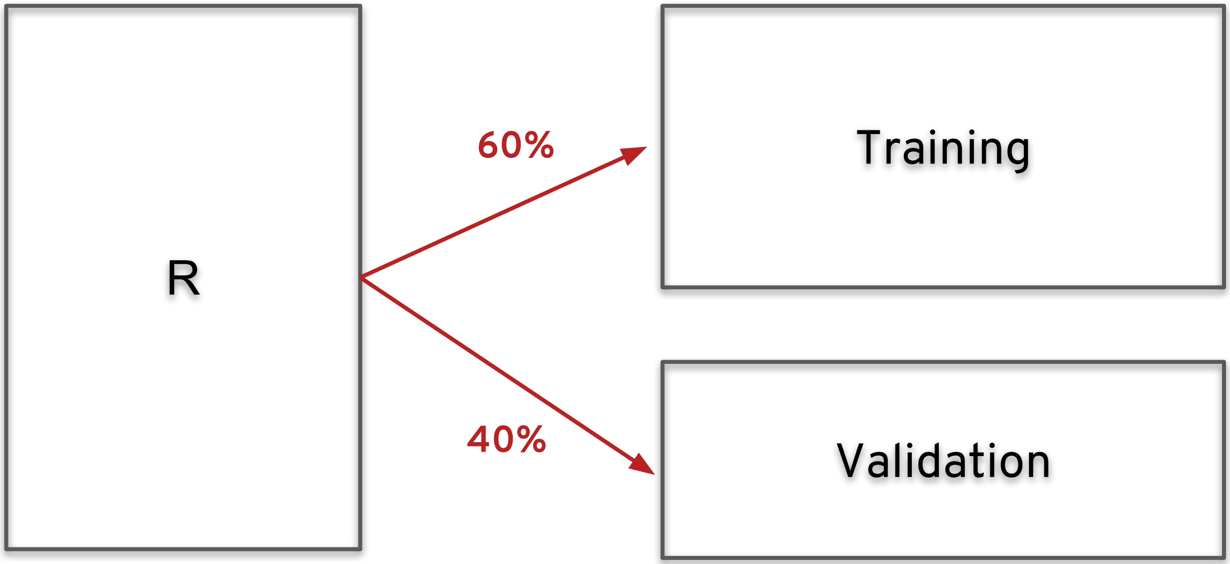
$$MSE = \frac{1}{N}\sum_{i=1}^{N}(f_i - y_i)^2$$

where $N$ is the number of data points,
$f_i$ the value returned by the model and
$y_i$ the actual value for data point $i$.

# Computing MSE

# Tuning Parameters



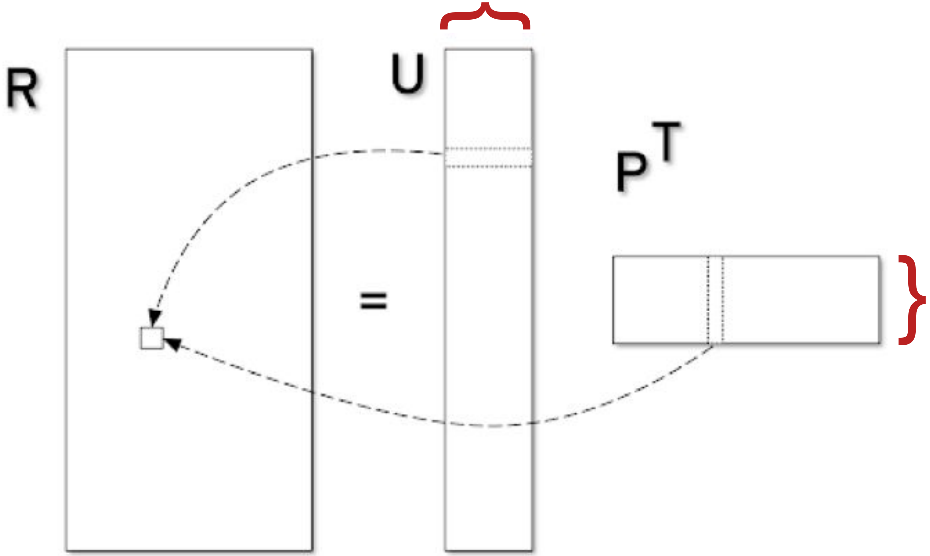



# Train the model

# Rank

## Dimension of the feature vectors

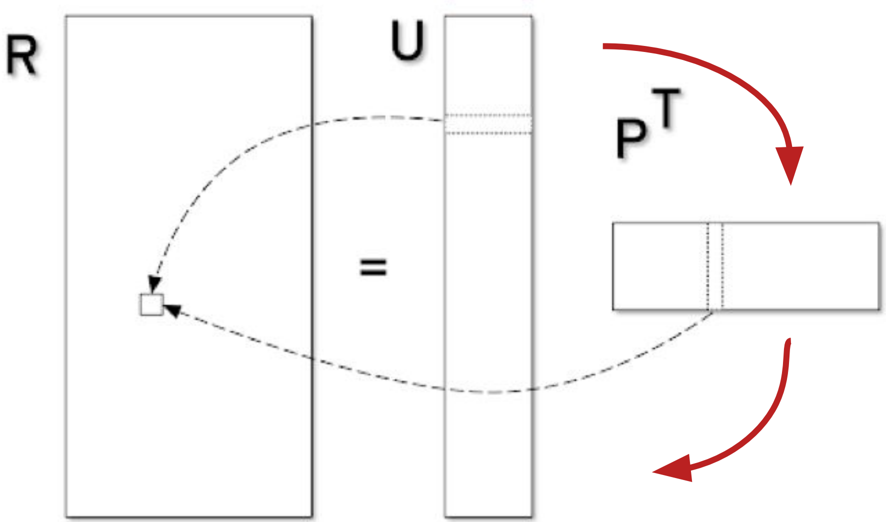
```
simple_als = ALS(maxIter=5, regParam=0.01, rank=3, coldStartStrategy='drop')
```



# maxIter

Number of optimisation steps implemented

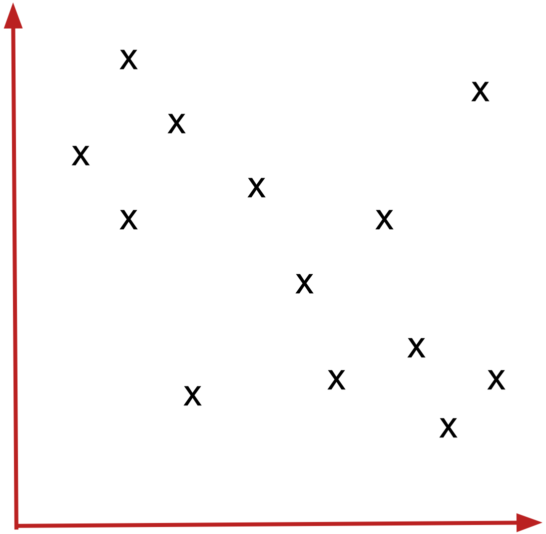
```
simple_als = ALS(maxIter=5, regParam=0.01, rank=3, coldStartStrategy='drop')
```



# regParam

Prevents overfitting the model

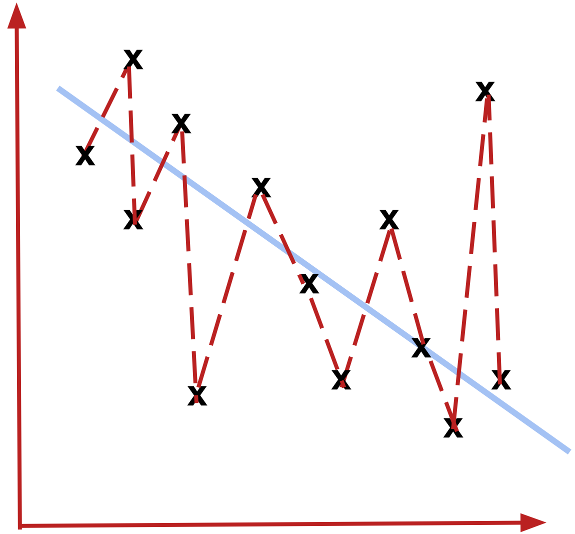
```
simple_als = ALS(maxIter=5, regParam=0.01, rank=3, coldStartStrategy='drop')
```



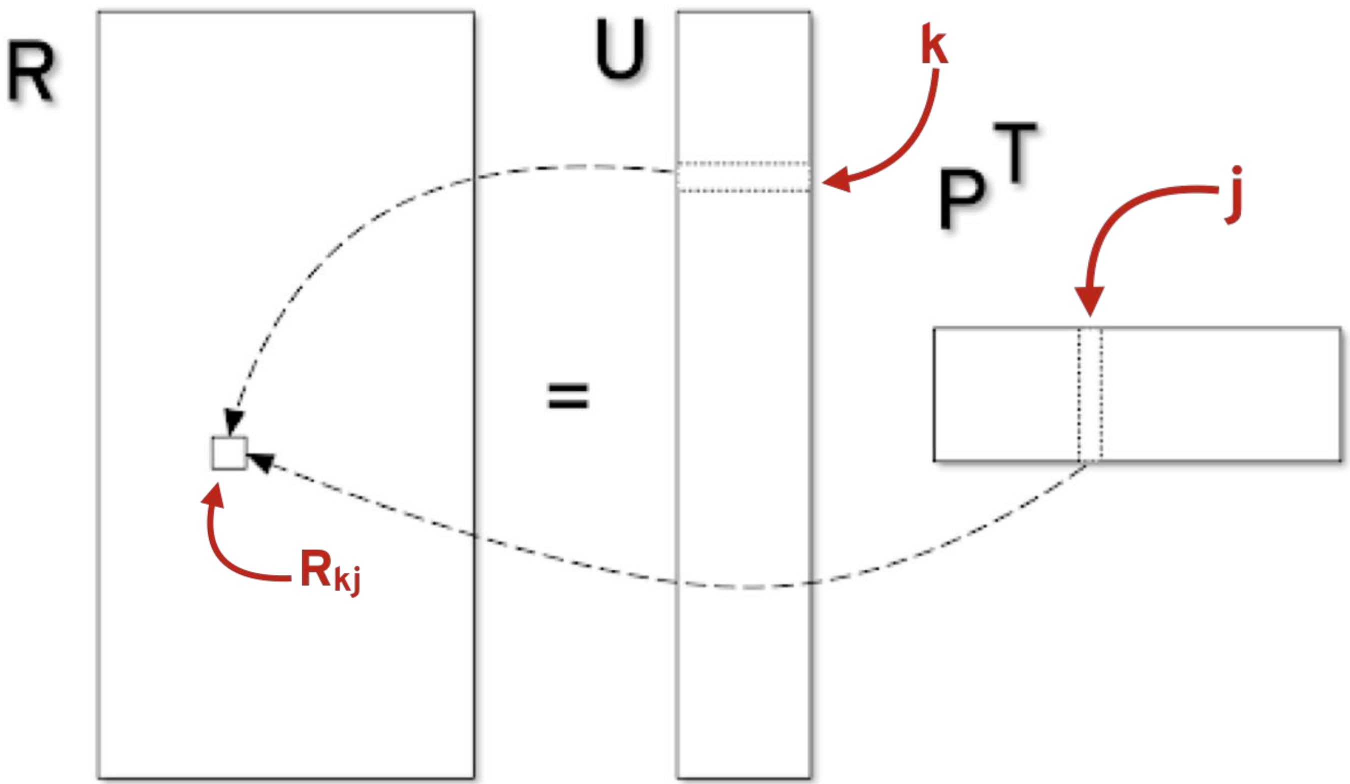
# regParam

Prevents overfitting the model

```
simple_als = ALS(maxIter=5, regParam=0.01, rank=3, coldStartStrategy='drop')
```

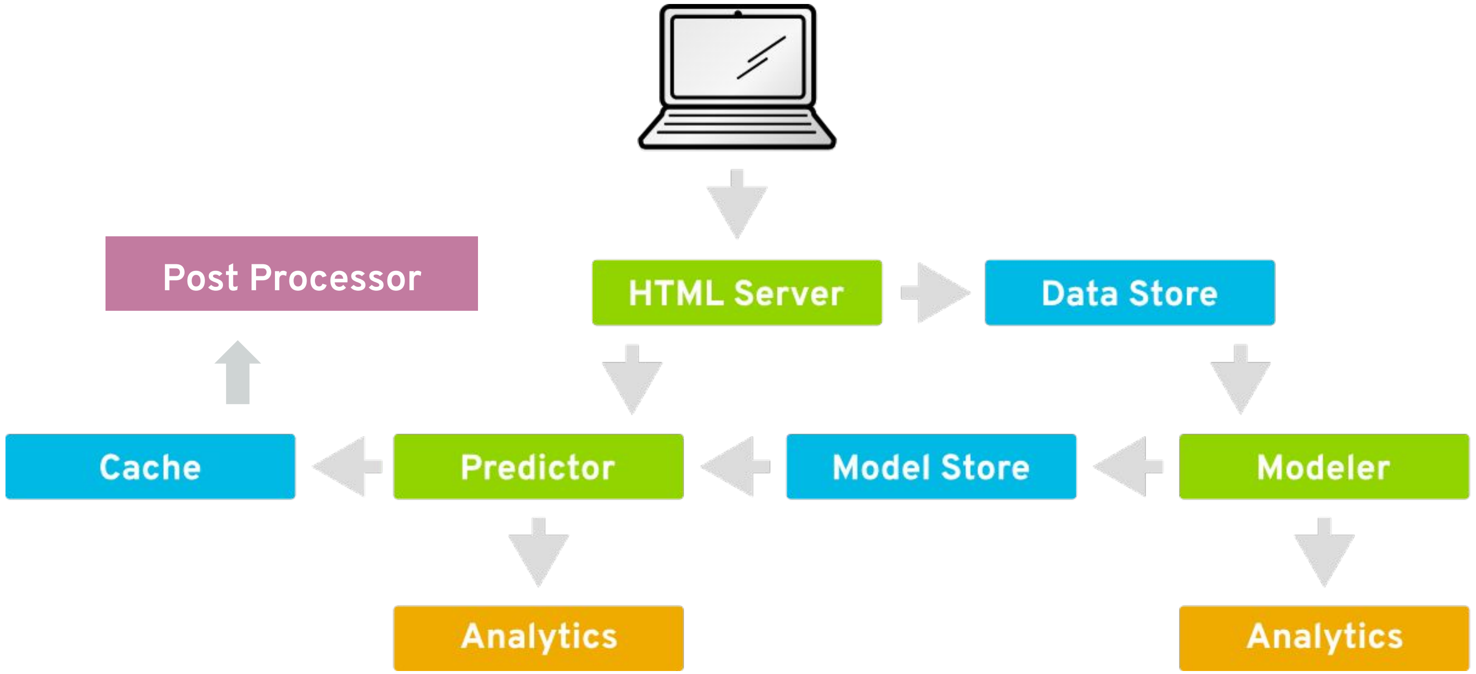


# Parameter Estimation

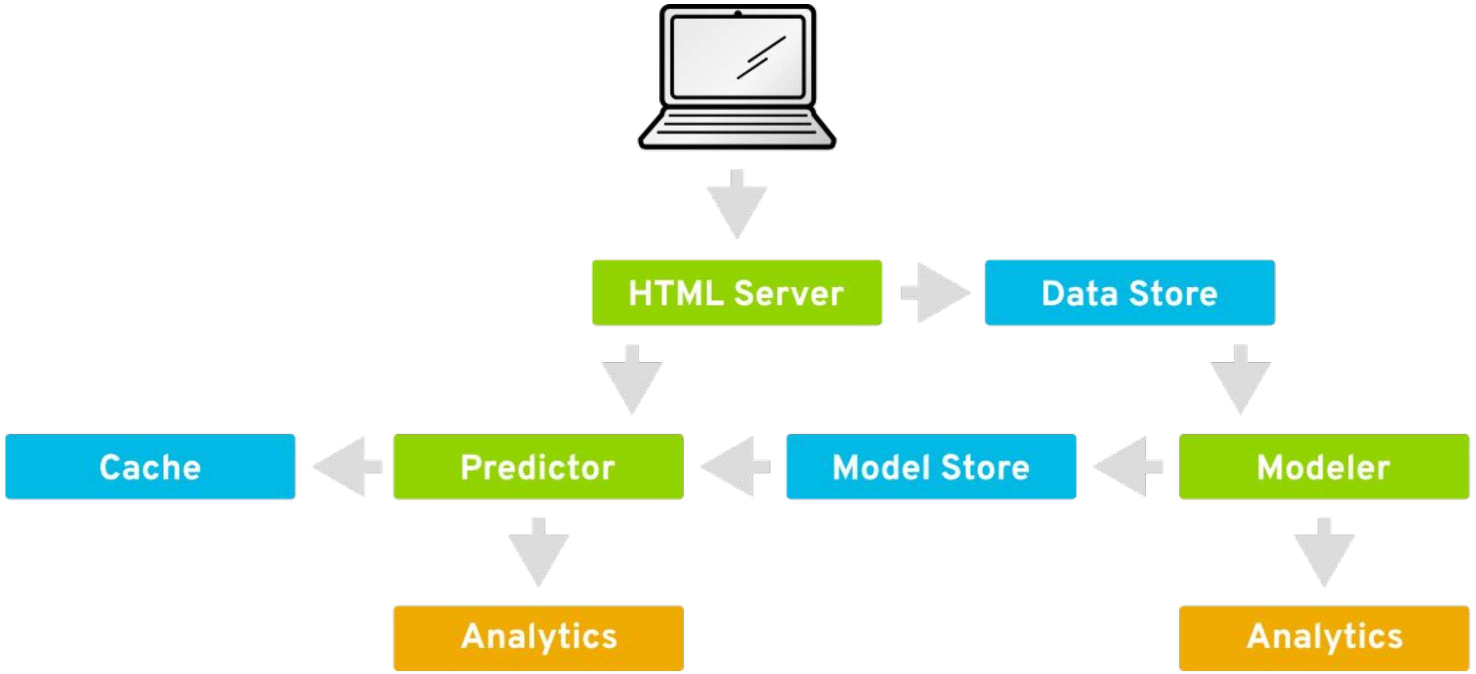

R
U
k
P
T
j
=
$R_{kj}$

# Making Predictions

# Post Processing



# Moving to a Production Environment



# RadAnalytics.io

rad analytics

Project Jiminy | Lightning Talk | Back to tutorials

Contents



## Recommendation engine service with Apache Spark

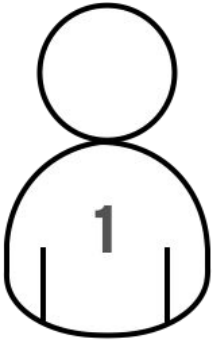
### Introduction

Project Jiminy is a service based application that implements a simple recommendation syster
alternating least squares methodology. That may sound complicated but through the source r
that creating a recommendation engine is more straightforward than expected.

With these instructions you will learn how to deploy Jiminy with the MovieLens dataset by the
represents a set of movies, users and their ratings of the movies. Although Jiminy uses this da
the services can be modified to utilize your own datasets.

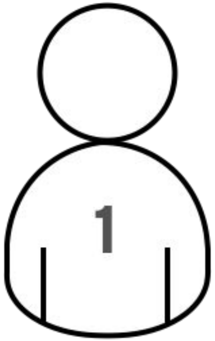
# Post Processing

# Implicit data



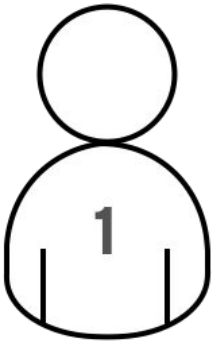

**Song A** **1 play**

# Implicit Data




| | |
|---|---|
| Song A | 1 play |
| Song B | 0 plays |

# Implicit Data




| | |
|---|---|
| Song A | 1 play |
| Song B | 0 plays |
| Song C | 100 plays |

# Preference and Confidence

$p_{ui} \in (0,1)$ **preference**

$r_{ui} \in \mathbb{R}$ **recording**

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases}$$

# Preference and Confidence

$$C_{ui} = 1 + \alpha r_{ui}$$

$p_{ui} \in (0,1)$ **preference**

$r_{ui} \in \mathbb{R}$ **recording**

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases}$$

# Preference and Confidence

**Minimisation:**

$$C_{ui}\left(p_{ui} - U_u X_i^T\right)$$

user vector

Item vector

$p_{ui} \in (0,1)$ **preference**

$r_{ui} \in \mathbb{R}$ **recording**

$$p_{ui} = \begin{cases} 1 & \text{if } r_{ui} > 0 \\ 0 & \text{if } r_{ui} = 0 \end{cases}$$

# ALS with Implicit Data

# Streaming Data

average rating          user bias          product bias

$$b_{x,y} = \mu + b_x + b_y$$

$$\hat{r}^{\star}_{x,y} = b_{x,y} + \hat{r}_{x,y}$$

# Streaming Data

**bias**

$$b_x \leftarrow b_x + \gamma\left(\epsilon_{x,y} - \lambda_x b_x\right)$$

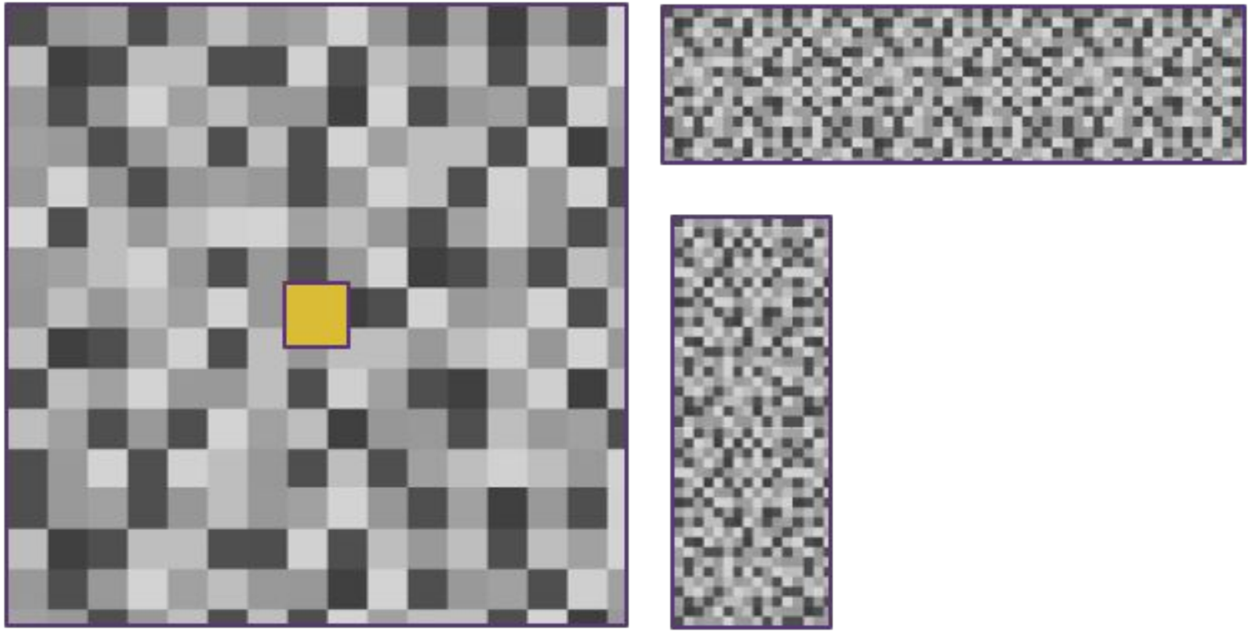
$$b_y \leftarrow b_y + \gamma\left(\epsilon_{x,y} - \lambda_y b_y\right)$$

**factors**

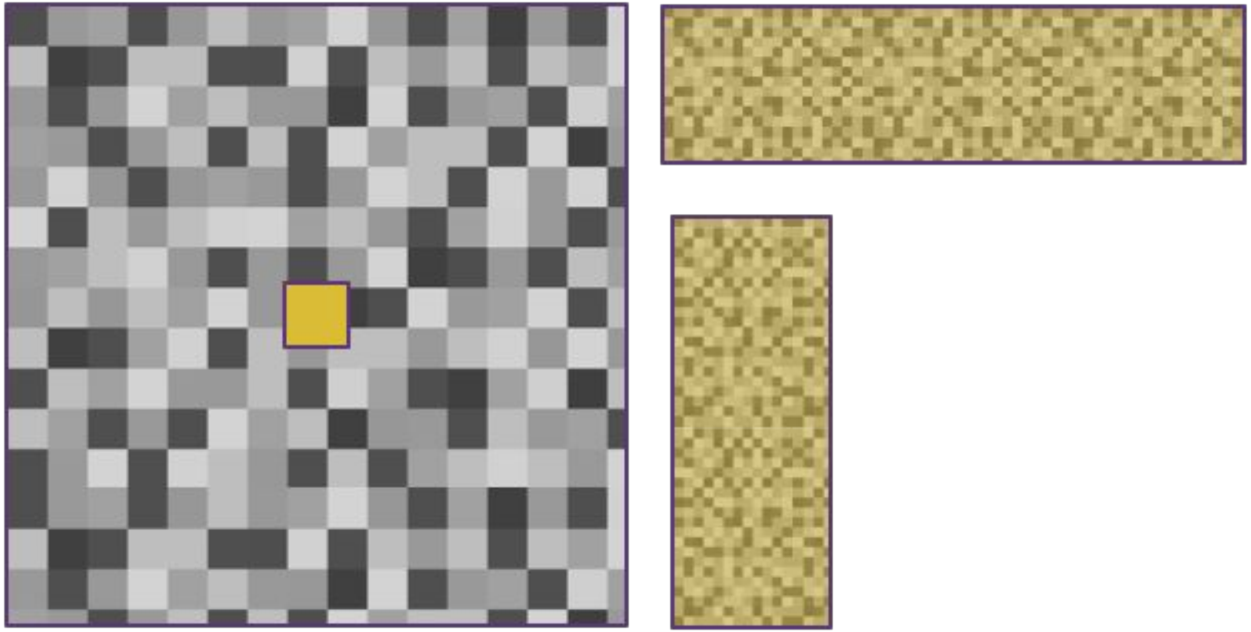
$$\mathsf{U}_x \leftarrow \mathsf{U}_x + \gamma\left(\epsilon_{x,y}\mathsf{P}_y - \lambda'_x \mathsf{U}_x\right)$$

$$\mathsf{P}_y \leftarrow \mathsf{P}_y + \gamma\left(\epsilon_{x,y}\mathsf{U}_x - \lambda'_y \mathsf{P}_y\right)$$

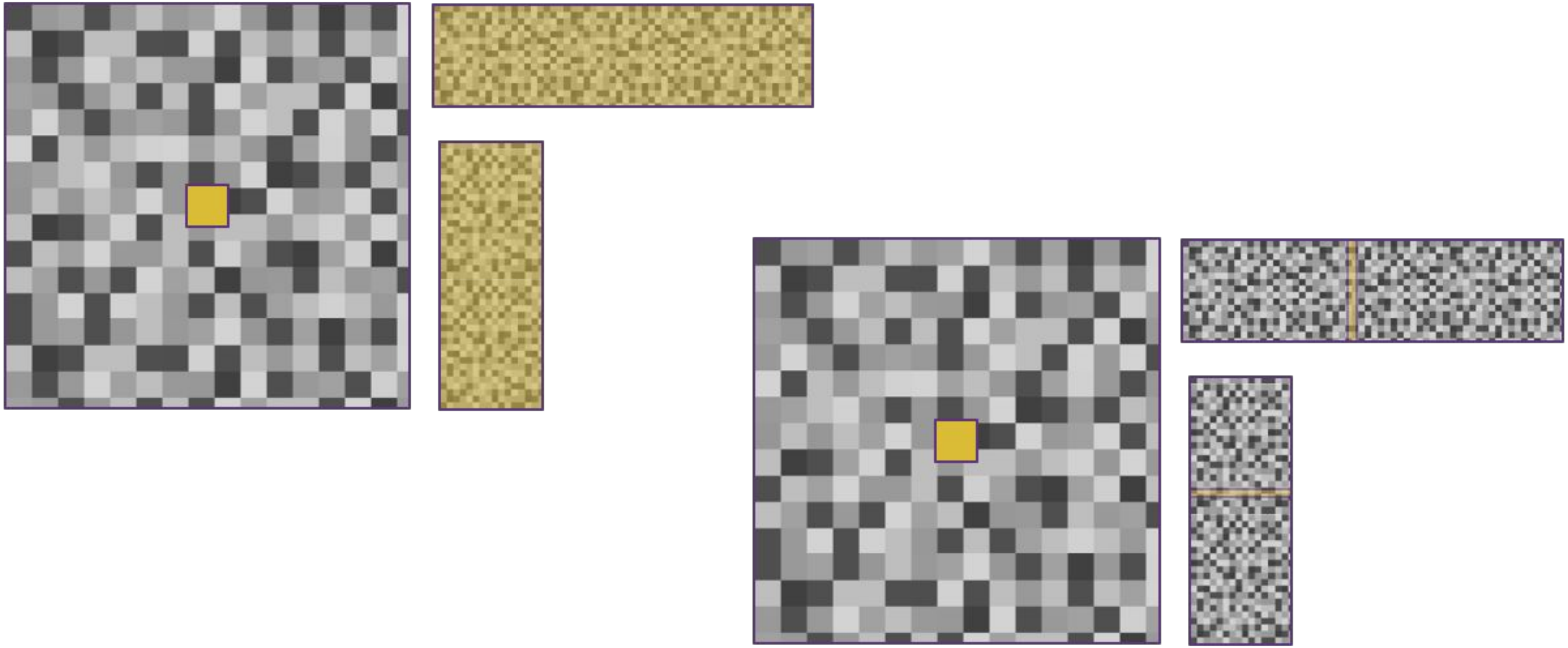
# Streaming *vs.* Batch



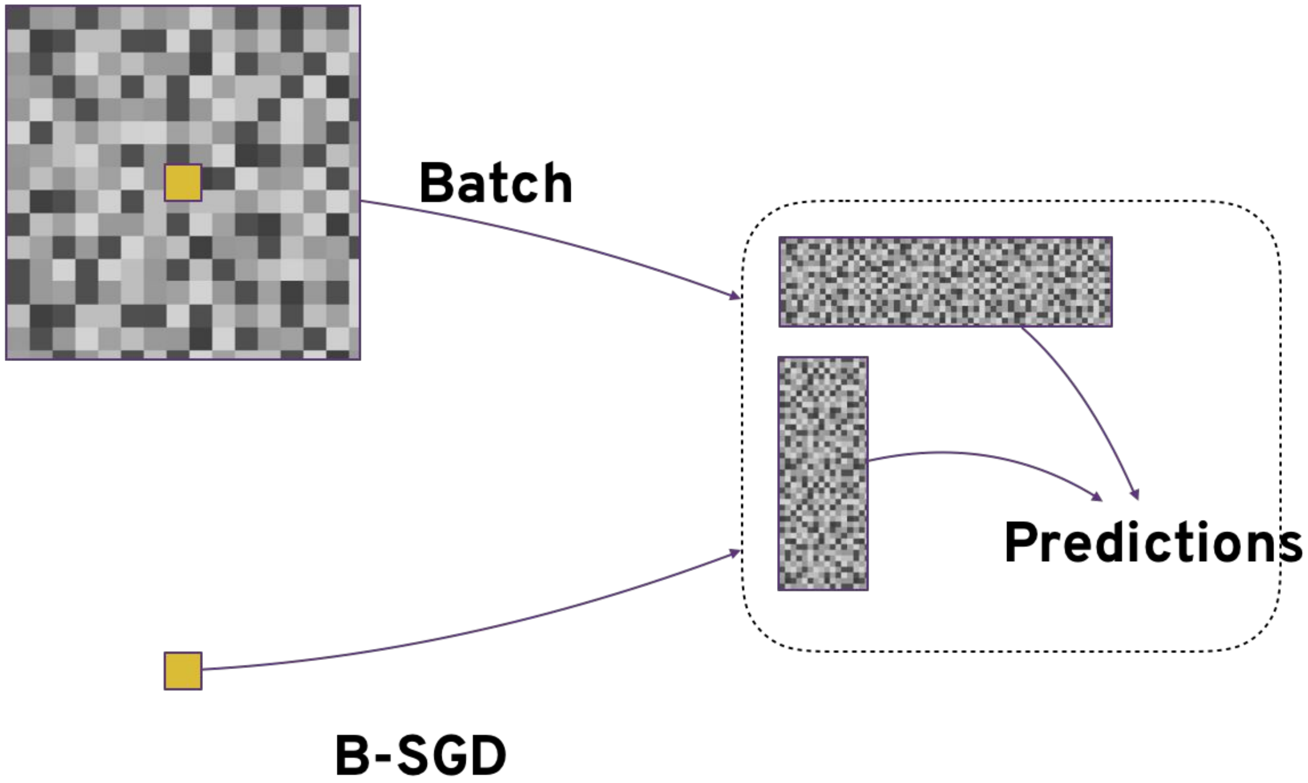
# Streaming *vs.* Batch



# Streaming *vs.* Batch



# Streaming *vs.* Batch



# Streaming Data

# Resources