Collaborative Filtering Microservices on Spark

Rui Vieira rcardoso@redhat.com
Sophie Watson sowatson@redhat.com

Overview

- What is ALS?
- Apache Spark
  - What does Spark already offer?
- Architecture
- Take aways

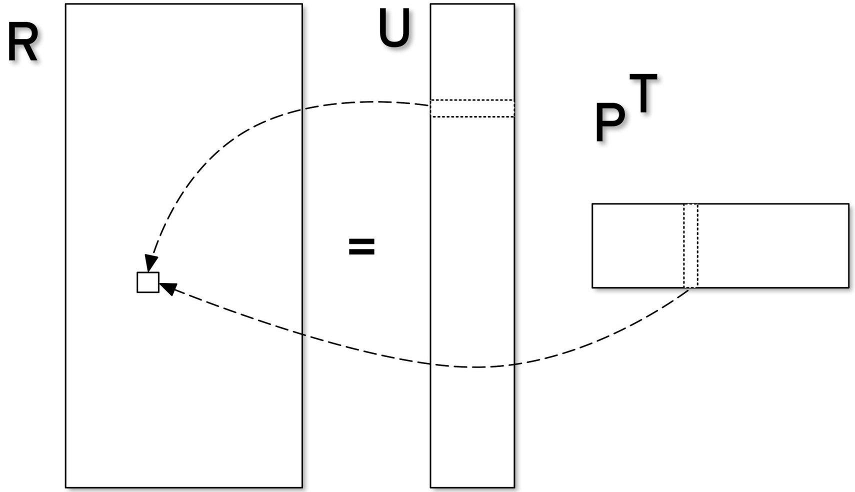
Collaborative Filtering

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

What is ALS?

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

What is ALS?

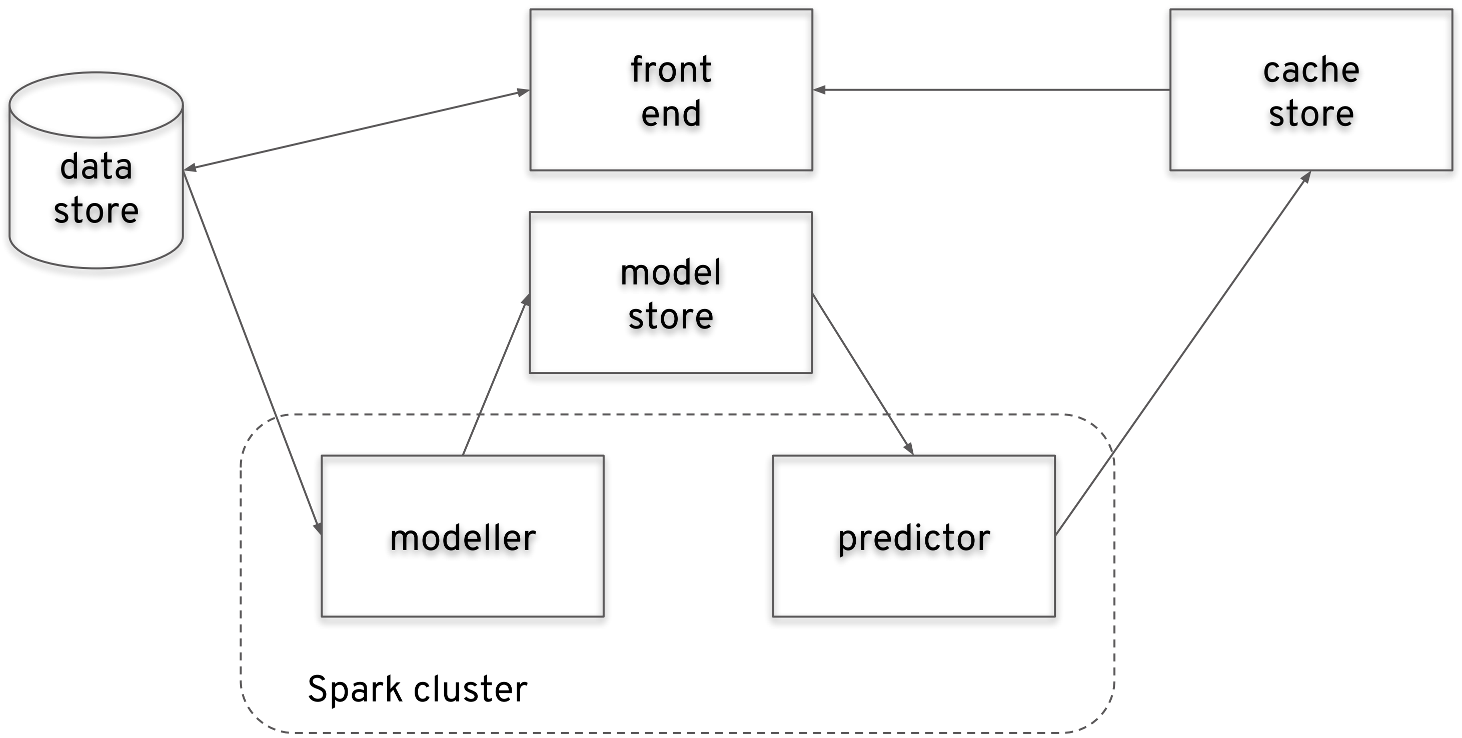


$$\hat{r}_{u,p} = U_u P_p^T$$

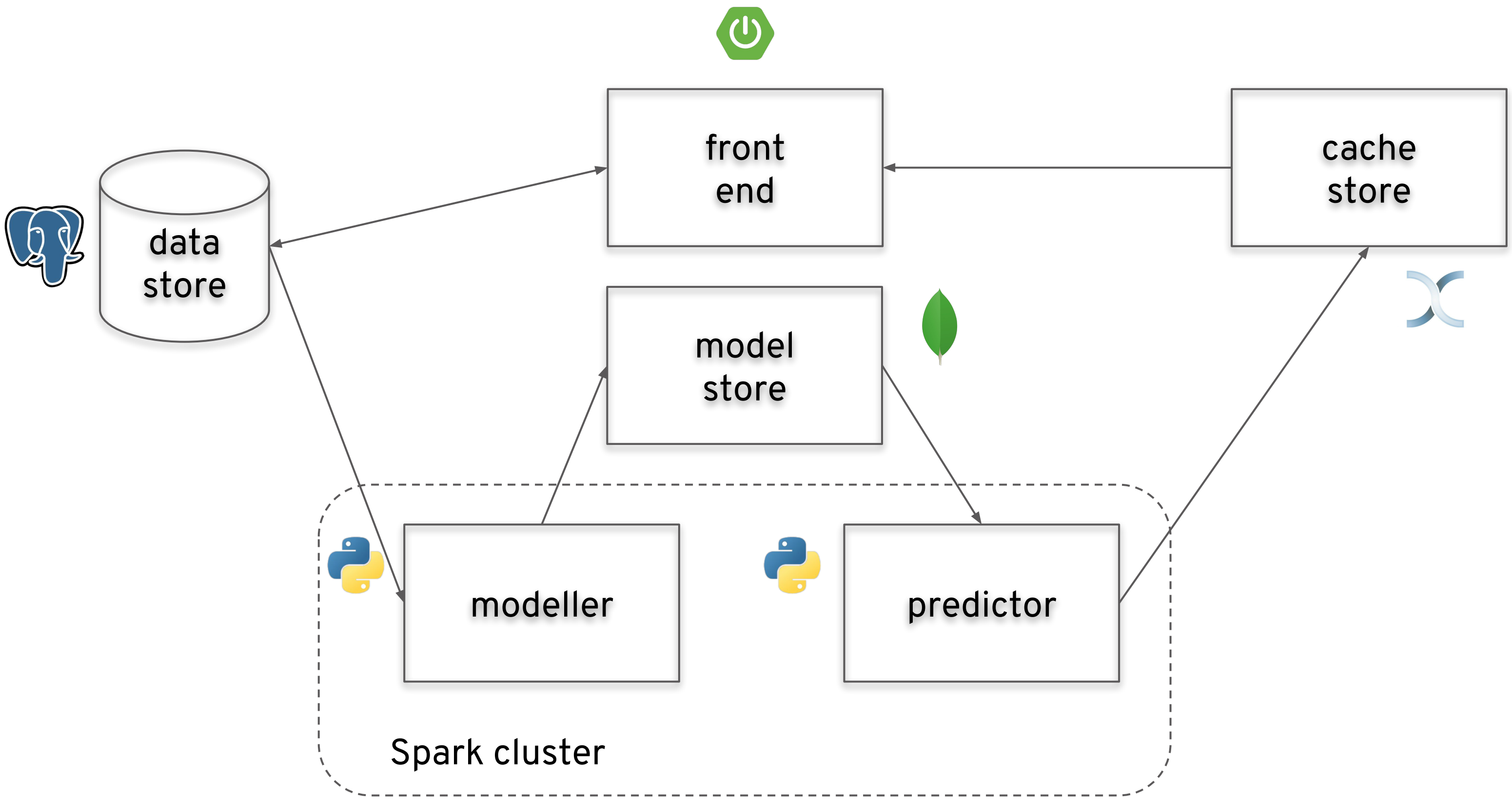
What is ALS?

$$R = \begin{array}{cc} \begin{array}{ccccc} \text{user 1} & \text{user 2} & \text{user 3} & \cdots & \text{user N} \end{array} & \\ \begin{bmatrix} 1 & 4.5 & 3.8 & \cdots & 3 \\ 3.2 & 3 & 3 & \cdots & 4 \\ 5 & 3 & 3.4 & \cdots & 3.1 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \cdots & 2.7 \end{bmatrix} & \begin{array}{c} \text{product 1} \\ \text{product 2} \\ \text{product 3} \\ \vdots \\ \text{product M} \end{array} \end{array}$$

Microservices

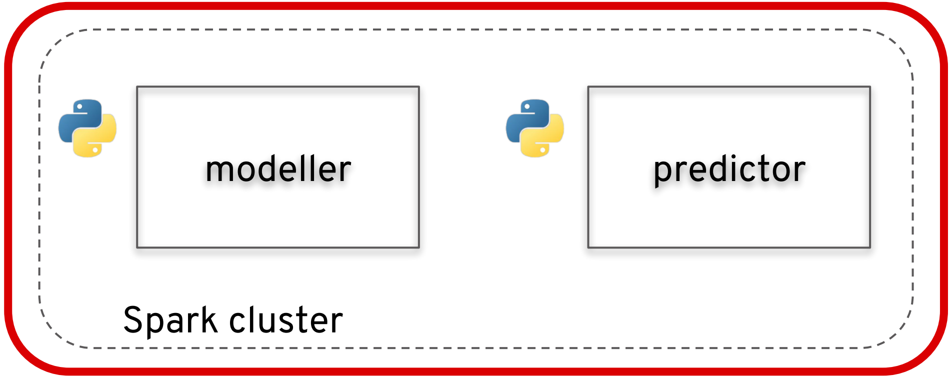


Microservices



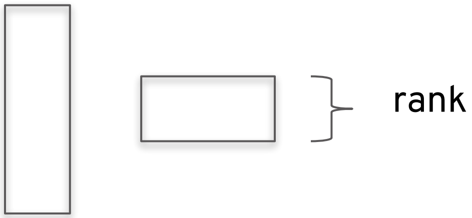
What does Spark offer?

```
from pyspark.mllib.recommendation import ALS
```



Modeller in Spark

$$\begin{bmatrix} 1 & 4.5 & ? & \cdots & 3 \\ ? & 3 & 3 & \cdots & 4 \\ 5 & 3 & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \cdots & ? \end{bmatrix} = R \approx U\,P^T =$$




```
model = ALS.train(ratings=ratings, rank=10, seed=42, iterations=10, lambda_=0.01)
```

Tuning Parameters

Rank

$$R \approx U\,P^{T}$$

$$p\left\{\overbrace{\begin{bmatrix} 1 & 4.5 & ? & \cdots & 3 \\ ? & 3 & 3 & \cdots & 4 \\ 5 & 3 & ? & \cdots & ? \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 2 & 4 & 1 & \cdots & ? \end{bmatrix}}^{n}\right. \Rightarrow p\left\{\underbrace{\begin{bmatrix} \quad & \quad \\ \quad & \quad \end{bmatrix}}_{k}\right. \times {\left.\underbrace{\begin{bmatrix} \quad & \quad \\ \quad & \quad \\ \quad & \quad \end{bmatrix}}_{k}\right\}n}^{T}$$

Predictor in Spark

```
unseen_prediction=model.predictAll(unseen)
```

output from ALS.train

(user, product) pairs

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/

Modelling

```
my_ratings = [(7451, 4.5), #mean girls
(1193, 5), #one flew over a ...
(96588, 4), #pitch perfect
(59725, 2), #satc
(78174, 1), #satc 2
(86833, 3), #bridesmaids,
```

```
my_ratings_rdd = sc.parallelize([(138493, r[0], r[1]) for r in my_ratings])

new_ratings = ratings.union(my_ratings_rdd)
```

```
model = ALS.train(ratings=new_ratings, rank=10, seed=42, iterations=10, lambda_=0.01)
```

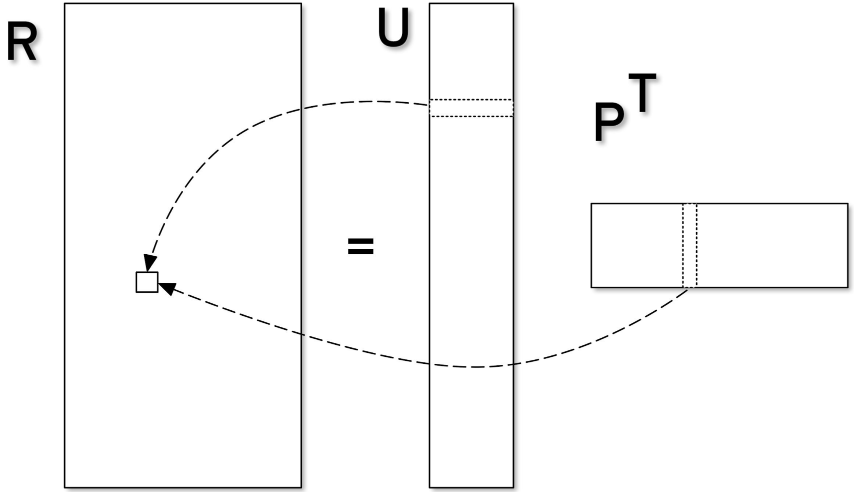
Prediction

```
to_predict_rdd = sc.parallelize([(138493, i) for i in range(11)])
predicted = model.predictAll(to_predict_rdd)
predicted.map(lambda x: (x[1], x[2])).join(movies).take(10)
```

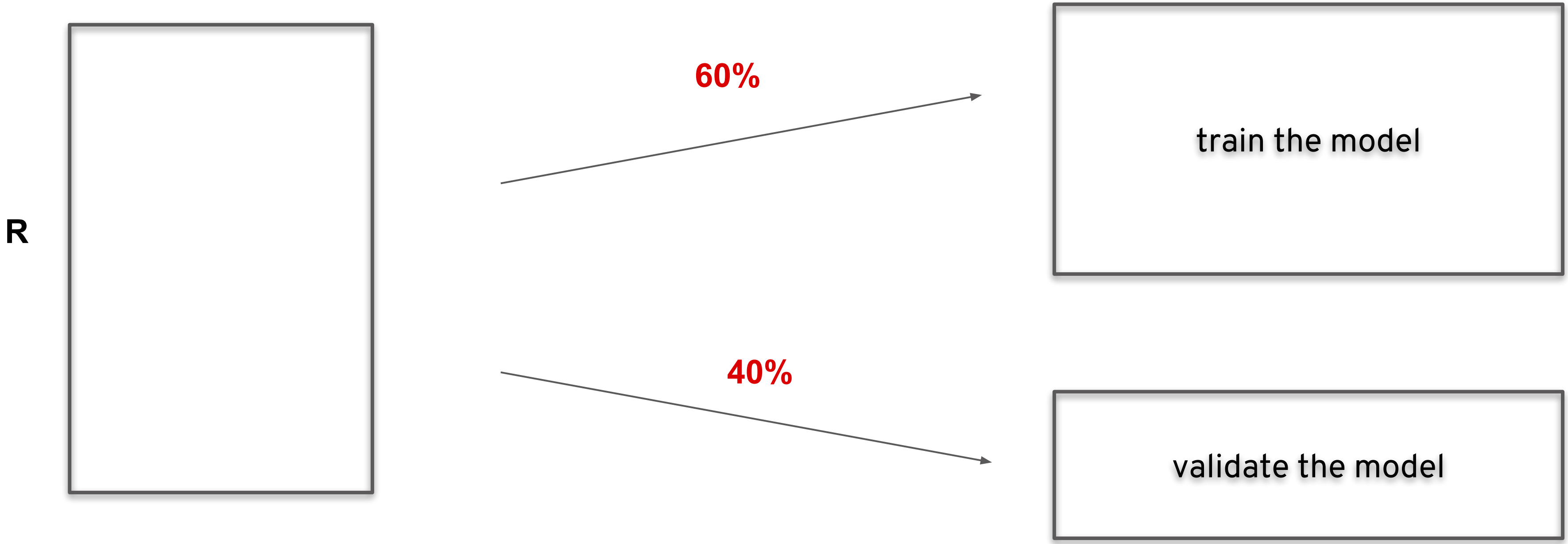
```
[(1, (4.064960525132482, u'Toy Story (1995)')),
 (2, (3.7083666928685886, u'Jumanji (1995)')),
 (3, (3.599853097386863, u'Grumpier Old Men (1995)')),
 (4, (2.945359779519419, u'Waiting to Exhale (1995)')),
 (5, (3.1136849725678406, u'Father of the Bride Part II (1995)')),
 (6, (4.535846503086157, u'Heat (1995)')),
 (7, (3.2605012167716203, u'Sabrina (1995)')),
 (8, (3.6625628164609014, u'Tom and Huck (1995)')),
 (9, (3.2067628803572914, u'Sudden Death (1995)')),
 (10, (3.9708589243819503, u'GoldenEye (1995)'))]
```

Iterative Tuning and Updating

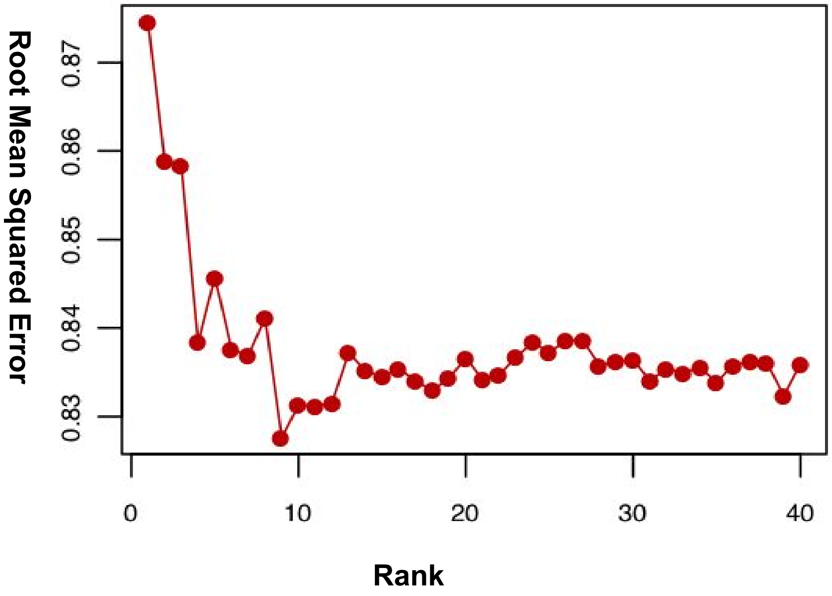
```
model = ALS.train(ratings=ratings, rank=10, seed=42, iterations=10, lambda_=0.01)
```

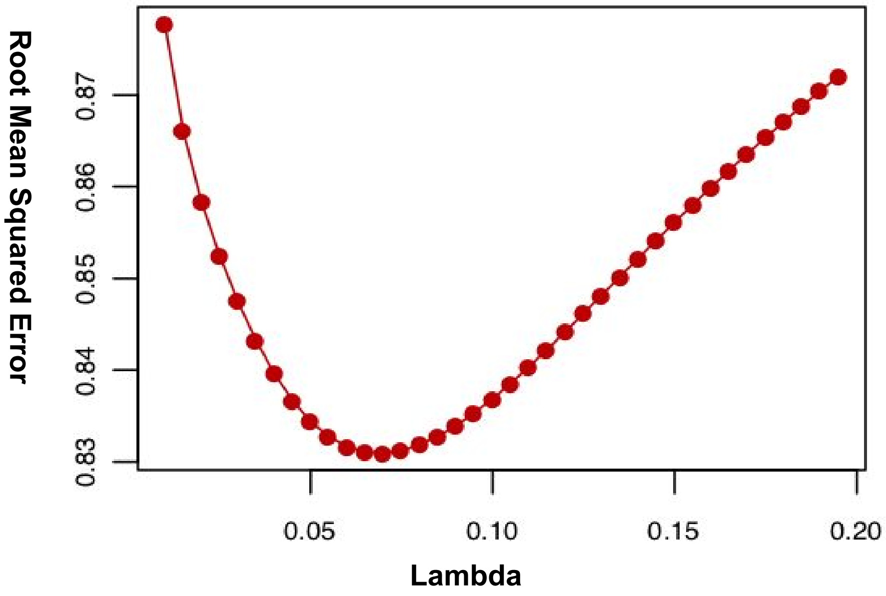


Training and Validation

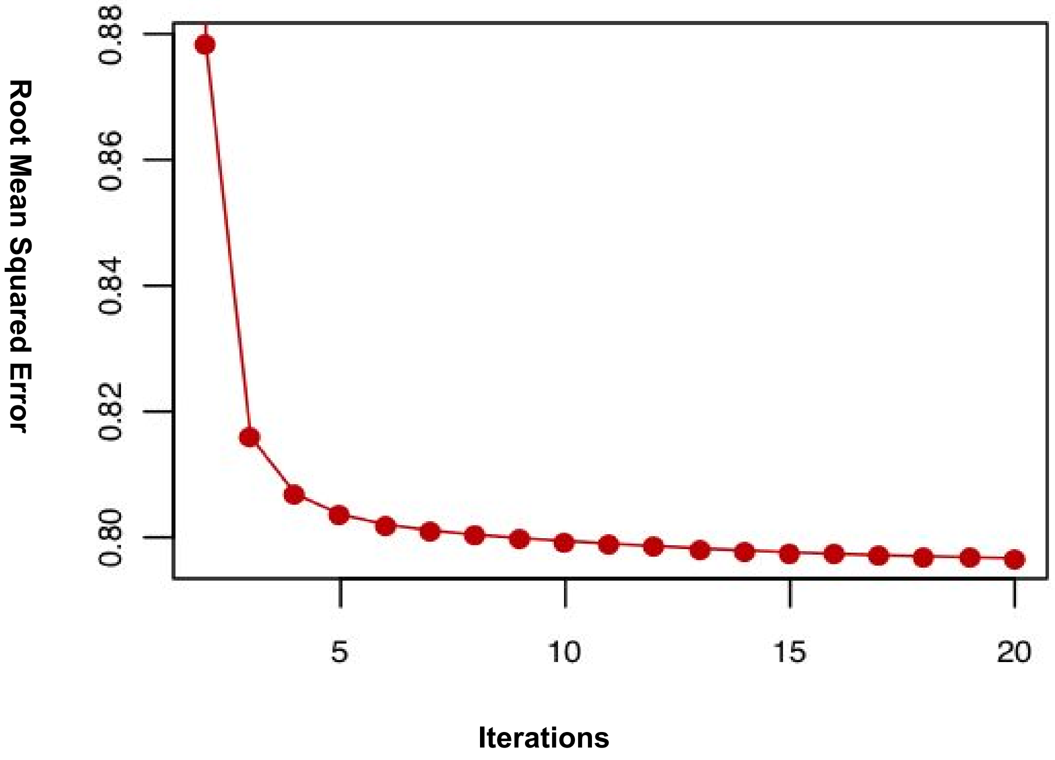


Rank and Lambda



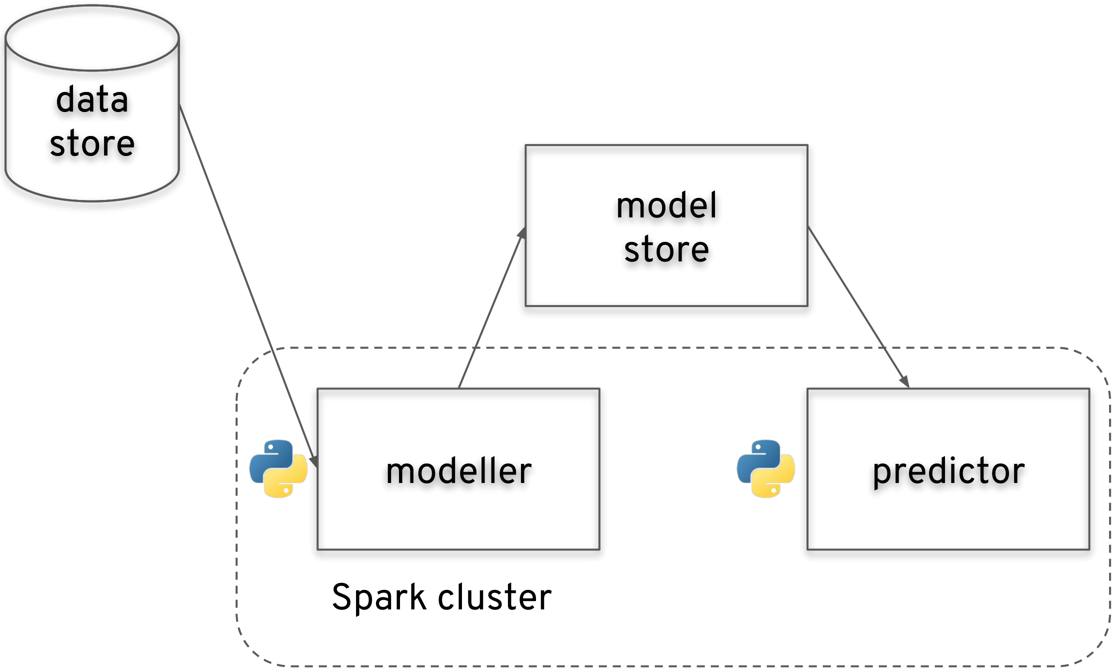


Iterations



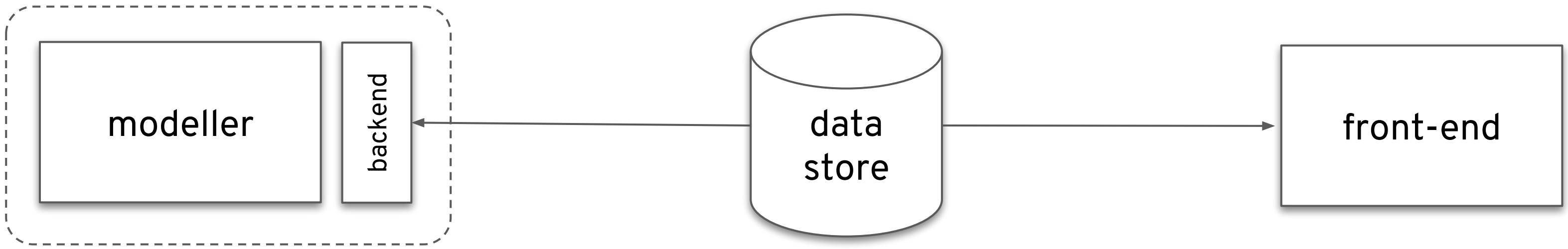
< 0.0001

Iterative updating of Model



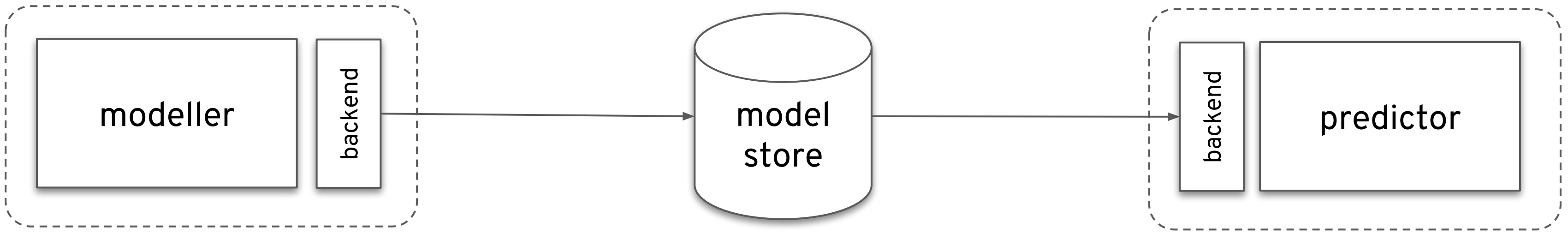
Data store

- PostgreSQL
- Users, products and ratings



Model Store

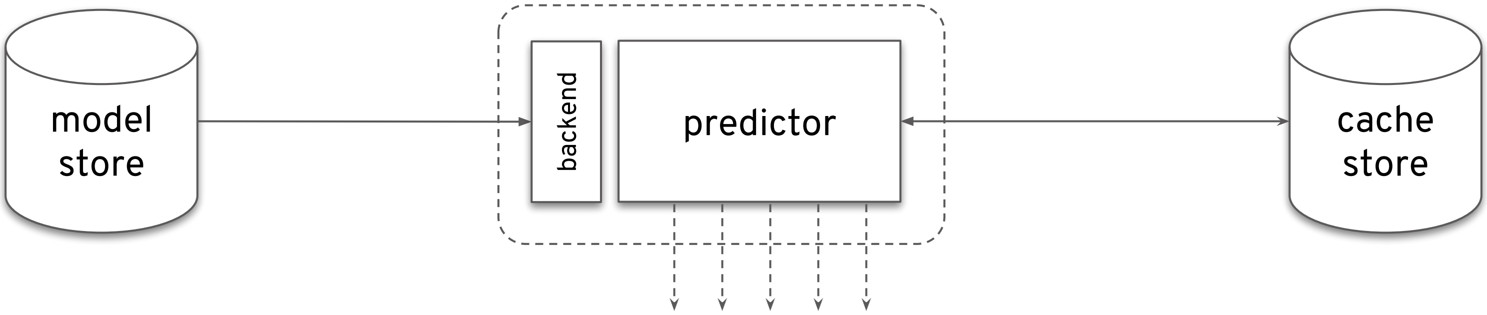
- Backend agnostic
- Store a representation of Spark's `MatrixFactorizationModel`



Predictor service

The `predictor` REST service

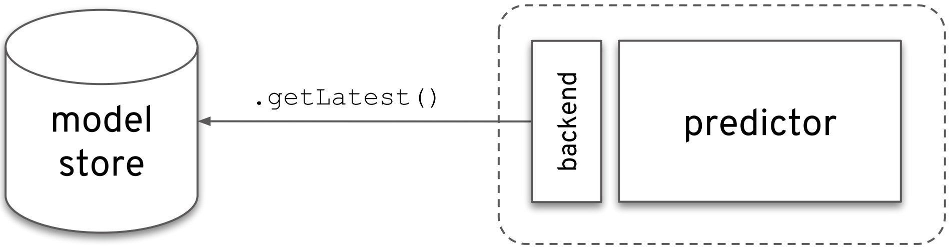
- Make predictions on a set `(user, product)` pairs
- Make rating top-k predictions (recommendations) to a `user`
- Connecting to the `model store` and loading models
- Populating the `cache store` with the latest predictions



Predictor service

The `predictor` initialization

- requests the latest model

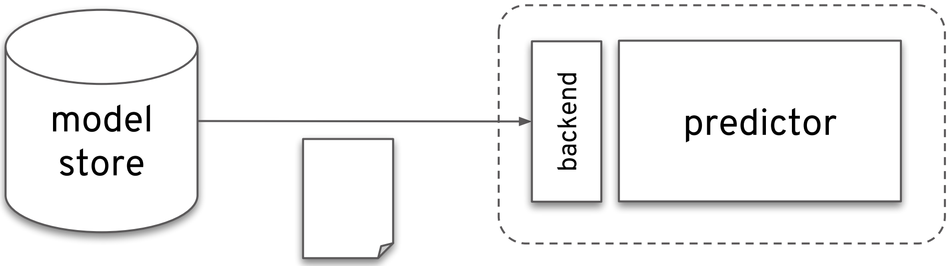


Predictor service

The `predictor` initialization

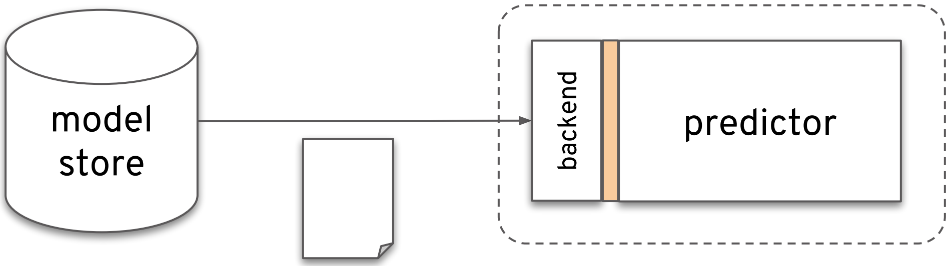
- requests the latest model
- MongoDB returns model as documents (metadata and latent factors)
- Spark ALS model instantiated as

```
MatrixFactorizationModel(uf: RDD[(Int, Array[Double])],
                         pf: RDD[(Int, Array[Double])])
```



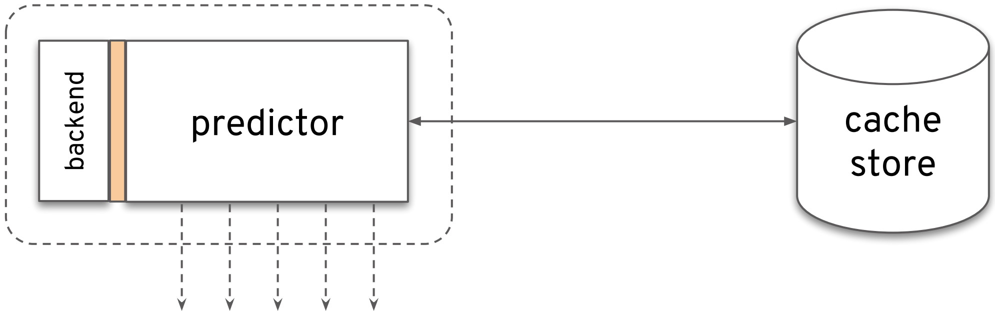
Predictor service

- Scala types - `RDD[(Int, Array[Double])]`
- Thin wrapper for type conversion
    - Python RDD → Scala RDD



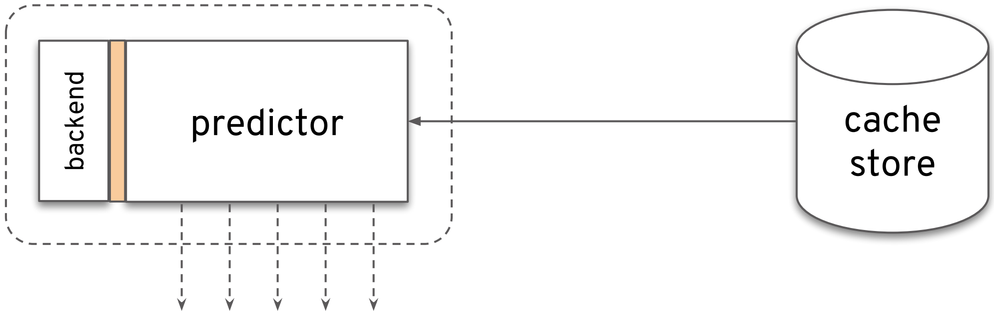
Predictor service

- REST endpoints
  - `POST /predictions/ratings`
  - `POST /predictions/rankings`
  - `GET /predictions/ratings/:id`
  - `GET /predictions/rankings/:id`



Predictor service

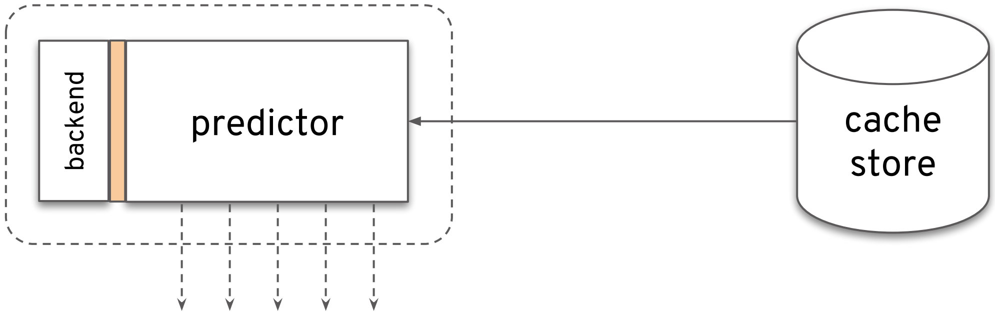
- REST endpoints
  - `POST /predictions/ratings`
  - `POST /predictions/rankings`
  - `GET /predictions/ratings/:id`
  - `GET /predictions/rankings/:id`



Predictor service

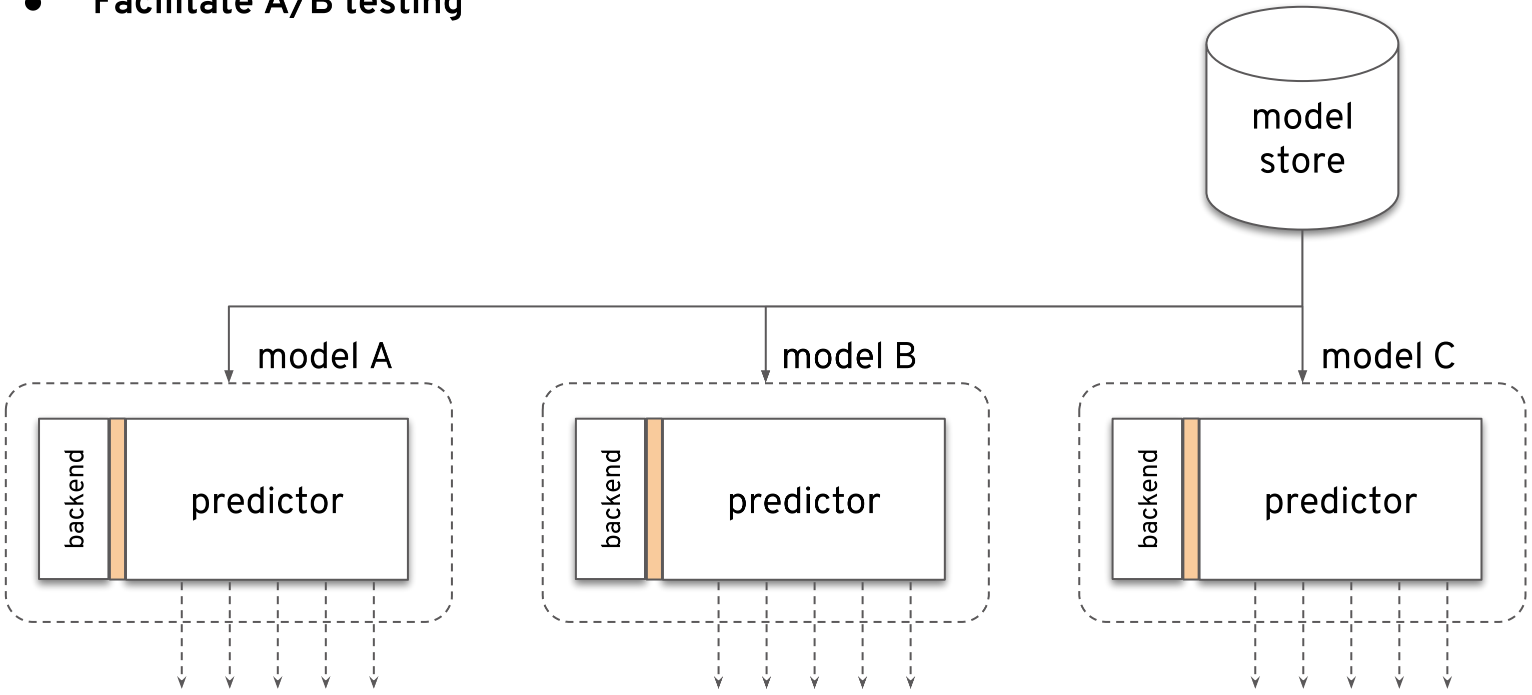
```
GET /predictions/ratings/b4ce … {
    "id": "b4ce3ba7132d49cf9d4024e40ff51162",
    "products": [
       {"id": 200, "rating": 3.172},
       {"id": 201, "rating": 3.268}
    ],
    "user": 100
}
```

```
GET /predictions/rankings/3efd … {
    "id": "3efd1646c7894d27abd48d4dc9497f47",
    "products": [
       {"id": 4518, "rating": 4.941},
       {"id": 4642, "rating": 4.571},
       …
    ],
    "topk": 20,
    "user": 100
}
```



Predictor service

- **Facilitate A/B testing**



Deploying on Openshift

Model store

```
oc new-app \
  -e MONGODB_USER=mongo \
  -e MONGODB_PASSWORD=mongo \
  -e MONGODB_DATABASE=models \
  -e MONGODB_ADMIN_PASSWORD=mongoadmin \
  --name mongodb \
  centos/mongodb-26-centos7
```

Deploying on Openshift

Predictor

```
oc new-app --template oshinko-pyspark-build-dc \
  -p GIT_URI=https://github.com/radanalyticsio/jiminy-predictor \
  -p SPARK_OPTIONS='--jars ./libs/spark-als-serializer_2.11-0.2.jar' \
  -e MODEL_STORE_URI=mongodb://mongo:mongo@mongodb/models \
  -p APP_FILE=app.py \
  -p APPLICATION_NAME=predictor
```

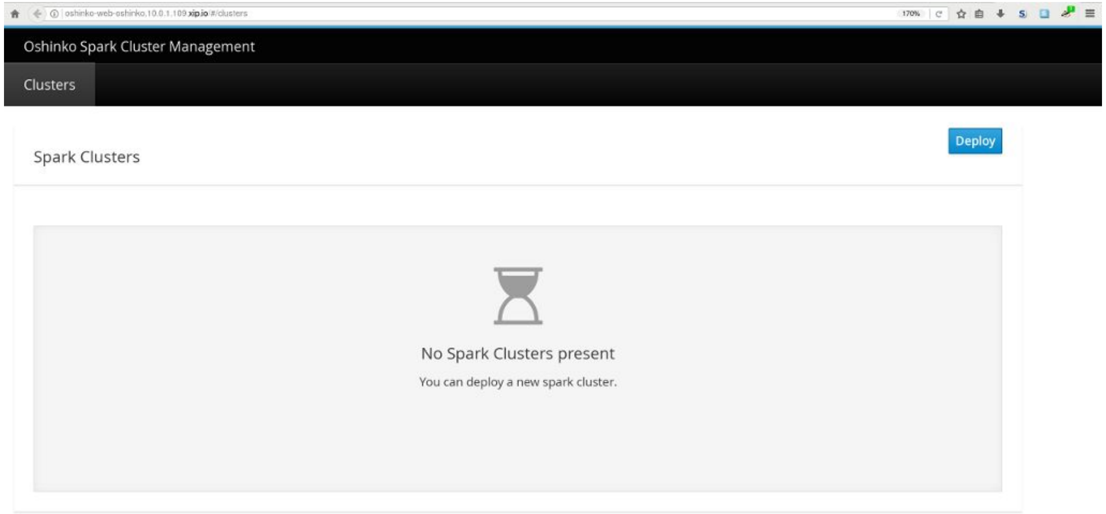
OPENSHIFT

Deploying on Openshift

Oshinko[1]

https://radanalytics.io/

```
oc create -f https://radanalytics.io/resources.yaml
```



[1] - https://radanalytics.io/

Takeaways

- Spark has all the stuff you need to make your own recommendation engine.
- By splitting your app into microservices you make a robust system.
- Easy to deploy app in containers.
- https://github.com/radanalyticsio (jiminy project)