cloudera

Streaming ML with Flink

Marton Balassi | Solutions Architect @ Cloudera*
@MartonBalassi | mbalassi@cloudera.com

Judit Feher | Data Scientist @ MTA SZTAKI
jfeher@ilab.sztaki.hu

*Work carried out while employed by MTA SZTAKI on the Streamline project.

This project has received funding from the European Union’s Horizon 2020 research and innovation program under grant agreement No 688191.

STREAMLINE.

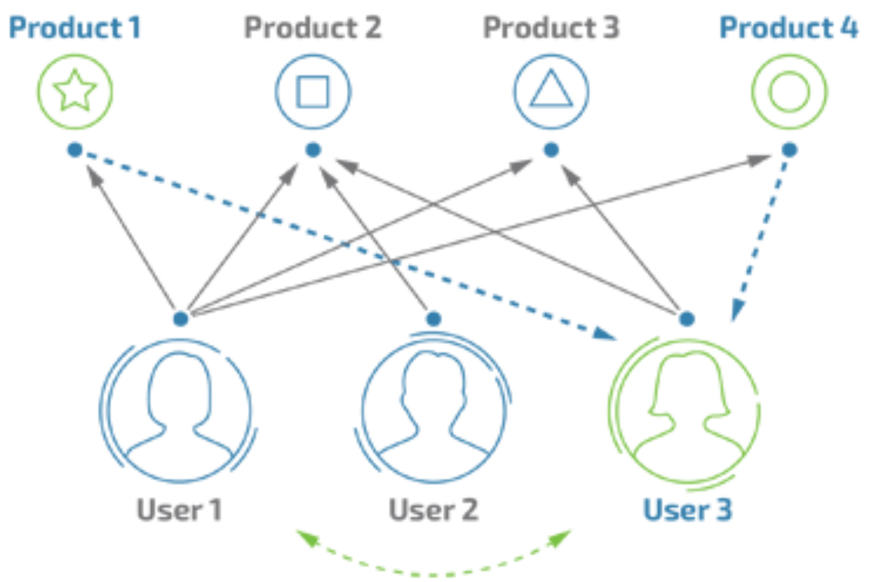
Outline

- Current FlinkML API through an example
- Adding streaming predictors
- Online learning
- Use cases in the Streamline project
- Summary

FlinkML example usage

Given a historical (training) dataset of user preferences
let us recommend desirable items for a set of users.

```
val env = ExecutionEnvironment.getExecutionEnvironment
val trainData = env.readCsvFile[(Int,Int,Double)](trainFile)
val testData = env.readTextFile(testFile).map(_.toInt)

val model = ALS()
        .setNumfactors(numFactors)
        .setIterations(iterations)
        .setLambda(lambda)
model.fit(trainData)

val prediction = model.predict(testData)
prediction.print()
```



Design motivated by the sci-kit learn API. More at http://arxiv.org/abs/1309.0238.

FlinkML example usage

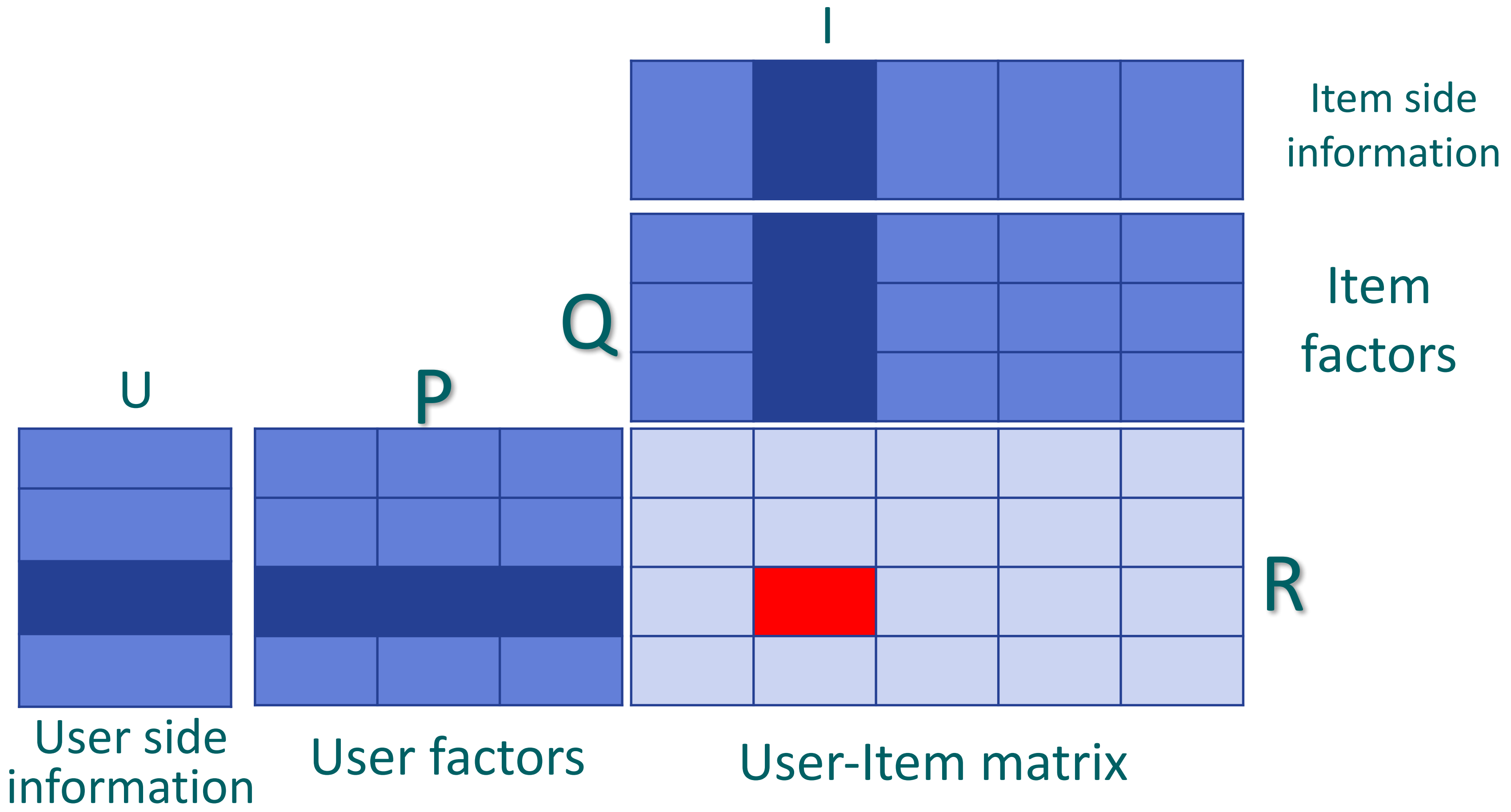
Given a historical (training) dataset of user preferences let us recommend desirable items for a set of users.

```
val env = ExecutionEnvironment.getExecutionEnvironment
val trainData = env.readCsvFile[(Int,Int,Double)](trainFile)
val testData = env.readTextFile(testFile).map(_.toInt)

val model = ALS()
        .setNumfactors(numFactors)
        .setIterations(iterations)
        .setLambda(lambda)
model.fit(trainData)

val prediction = model.test(testData)
prediction.print()
```

This is a batch input.
But does it need to be?

cloudera

A little recommender theory



- R is potentially huge, approximate it with P∗Q
- Prediction is TopK(user's row ∗ Q)

Prediction is a natural fit for streaming.

A closer (schematic) look at the API

A DataSet and a record level API to implement the algorithm
(Prediction is always done on a model already trained)

```
trait PredictDataSetOperation[Self, Testing, Prediction] {
        def predictDataSet(instance: Self, input: DataSet[Testing]) : DataSet[Prediction]
}

trait PredictOperation[Instance,  Model, Testing, Prediction] {

        def getModel(instance: Instance) : DataSet[Model]

        def predict(value: Testing, model: Model) : DataSet[Prediction]
}
```

The record level version is arguably more convenient
It is wrapped into a default dataset level implementation

A closer (schematic) look at the API

Three well-picked traits go a long way

```
trait Estimator {
        def fit[Training](training: DataSet[Training])(implicit f: FitOperation[Training]) = {
                f.fit(training)
        }
}

trait Transformer extends Estimator {
        def transform[I,O](input: DataSet[I])(implicit t: TransformDataSetOperation[I,O]) = {
                t.transform(input)
        }
}

trait Predictor extends Estimator {
        def predict[Testing](testing: DataSet[Testing])(implicit p: PredictDataSetOperation[T]) = {
                p.predict(testing)
        }
}
```

cloudera

Could we share the model with a streaming job?

Learn in batch, predict in streaming

```scala
val env = ExecutionEnvironment.getExecutionEnvironment
val strEnv = StreamExecutionEnvironment.getExecutionEnvironment
val trainData = env.readCsvFile[(Int,Int,Double)](trainFile)
val testData = env.socketTextStream(...).map(_.toInt)

val model = ALS()
        .setNumfactors(numFactors)
        .setIterations(iterations)
        .setLambda(lambda)
model.fit(trainData)

val prediction = model.predictStream(testData)
prediction.print()
```

cloudera

A closer (schematic) look at the streaming API

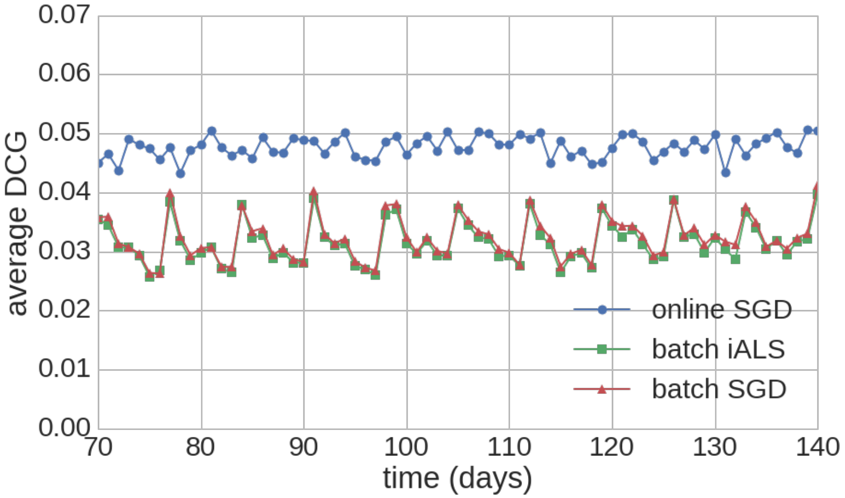
- Implicit conversions from the batch Predictors to StreamPredictors
- The model is stored then loaded into a stateful RichMapFunction processing the input stream
- Default wrapper implementations to support both the DataStream level and the record level implementations
- Adding the streaming predictor implementation for an algorithm given the batch one is trivial

```
trait PredictDataSetOperation[Self, Testing, Prediction] {
        def predictDataSet(instance: Self, input: DataSet[Testing]) : DataSet[Prediction]
}

trait PredictDataStreamOperation[Self, Testing, Prediction] {
        def predictDataStream(instance: Self, input: DataStream[Testing]) : DataStream[Prediction]
}
```

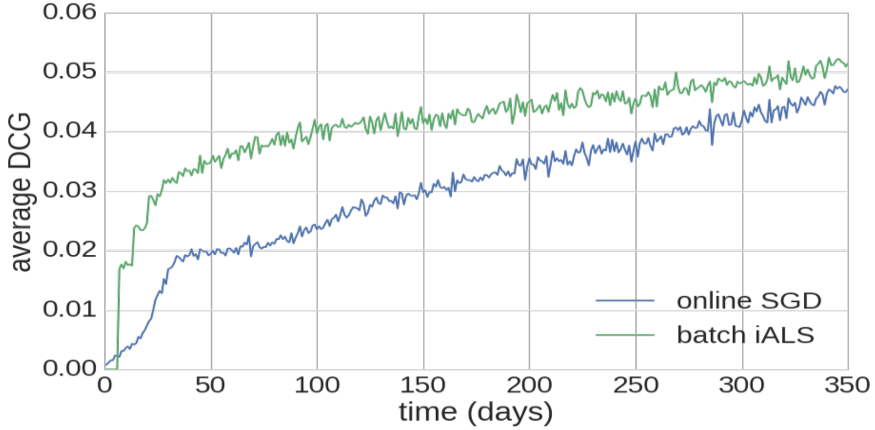
cloudera

Recommender systems in batch vs online learning

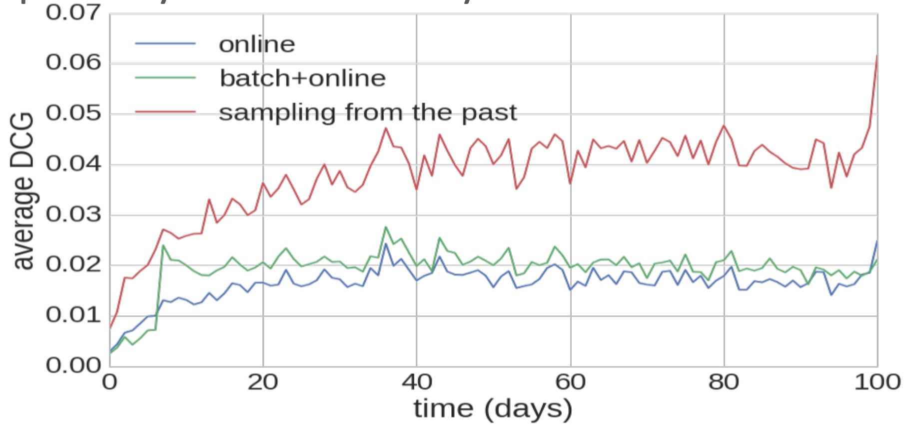
- “30M” Music listening dataset crawled by the CrowdRec team
- Implicit, timestamped music listening dataset
- Each record contains:
  [timestamp, user , artist, album, track, …]
- We always recommend and learn when the user interacts with an item at the *first time*
- ~50,000 users, ~100,000 artists, ~500,000 tracks



- This happens when we shuffle the time



- A partially batch online system



cloudera

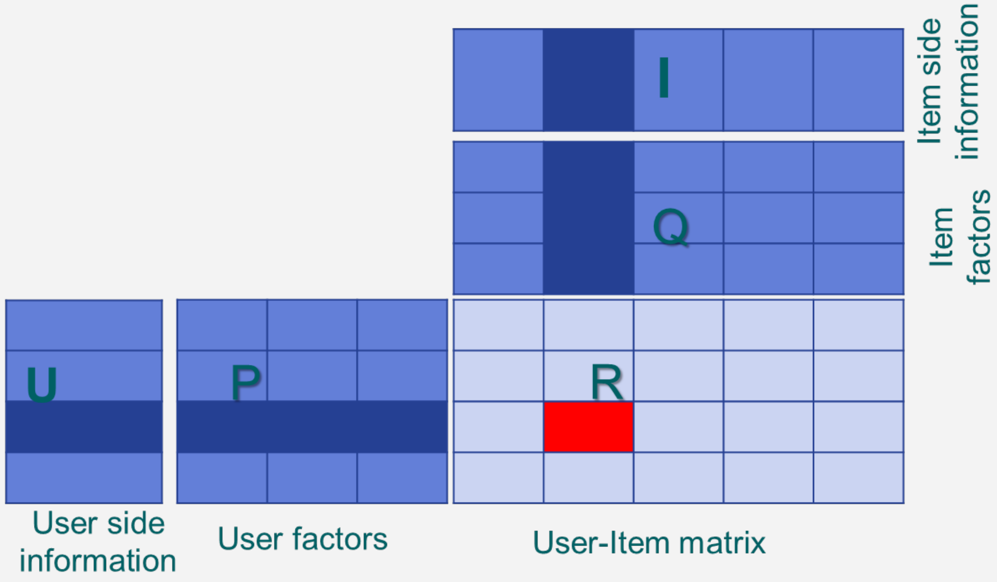
Use cases in the Streamline project

Judit Fehér
Hungarian Academy of Sciences

How iALS works and why is it different from ALS

ALS Problem to solve: $R_i = P^T Q_i$

– Linear regression



Error function

$$L = \left\| R - \hat{R} \right\|_{frob}^{2} + \lambda_U \| P \|_{frob}^{2} + \lambda_I \| Q \|_{frob}^{2}$$

Implicit error function

$$L = \sum_{u=1,i=1}^{S_U,S_I} w_{u,i} \left(\hat{r}_{u,i} - r_{u,i} \right)^2 + \lambda_U \sum_{u=1}^{S_U} \| P_u \|^2 + \lambda_I \sum_{i=1}^{S_I} \| Q_i \|^2$$

- Weighted MSE
- $w_{u,i} = \begin{cases} w_{u,i} & \text{if } (u,i) \in T \\ w_0 & \text{otherwise} \end{cases}$ $\quad w_0 \ll$
  $w_{u,i}$
- Typical weights:
  $w_0 = 1, \quad w_{u,i} = 100 * supp(u,i)$
- What does it mean?
  - Create two matrices from the events
  - (1) Preference matrix
    - Binary
    - 1 represents the presence of an event
  - (2) Confidence matrix
    - Interprets our certainty on the corresponding values in the first matrix
    - Negative feedback is much less certain

Machine learning: batch, streaming? Combined?

Batch recommender

- Repeatedly read all training data multiple times
- Stochastic gradient: use multiple times in random order
- Elaborate optimization procedures, e.g. SVM

+ More accurate (?)

+ Easy to implement (?)

Streaming recommeder

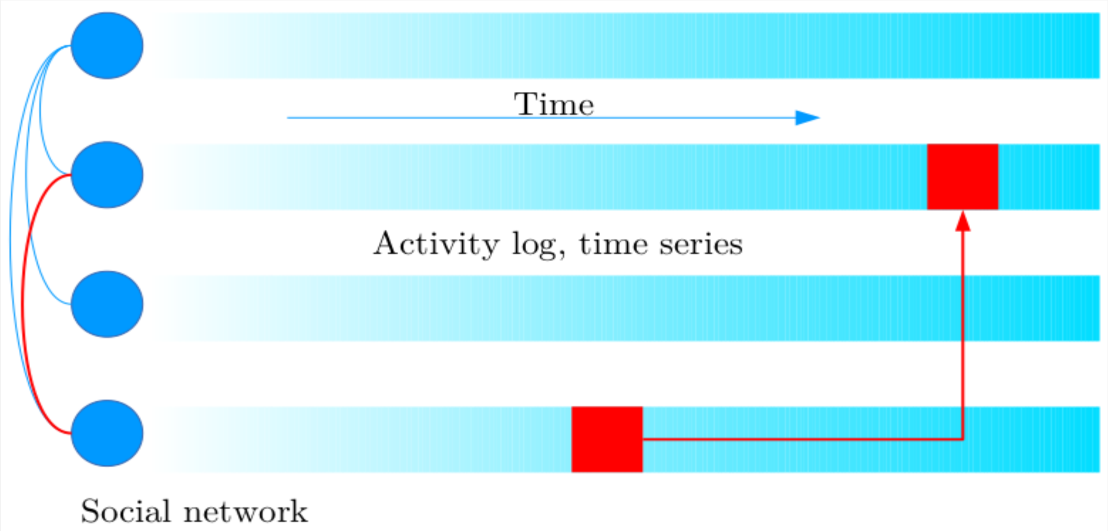
- Online learning
- Update immediately, e.g. with large learning rate
- Data streaming
- Read training/testing data only once, no chance to store
- Real time / Interactive

+ More timely, adapts fast

- Challenging to implement

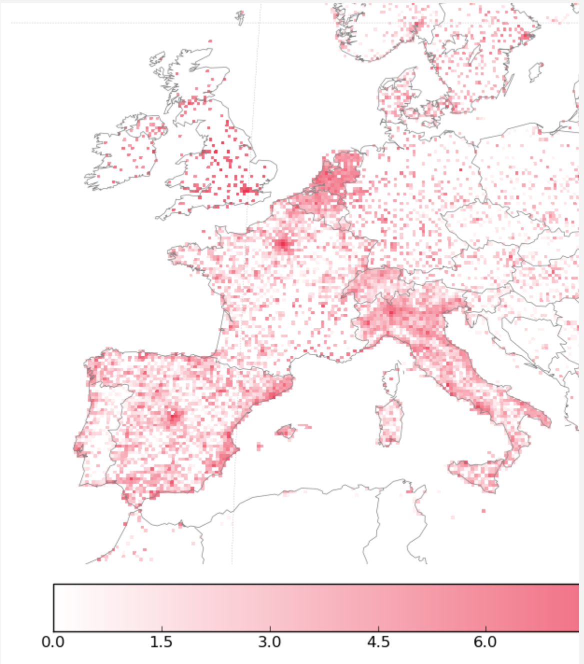
Contextualized recommendation (NMusic)

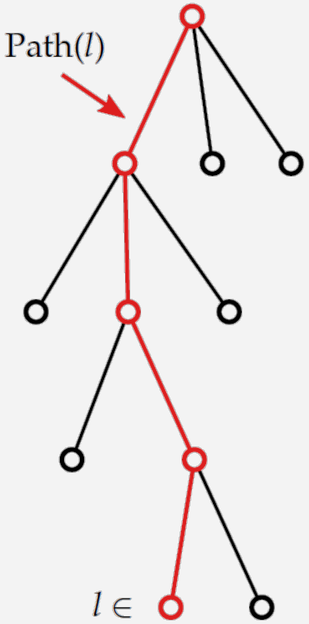
Social recommendation



R.Palovics, A.A.Benczur, L.Kocsis, T.Kiss, E.Frigo. "Exploiting temporal influence in online recommendation", ACM RecSys (2014)

Geo recommendation





Palovics, Szalai, Kocsis, Pap, Frigo, Benczur. „Location-Aware Online Learning for Top-k Hashtag Recommendation”, LocalRec (2015)

Internet Memory Research use cases

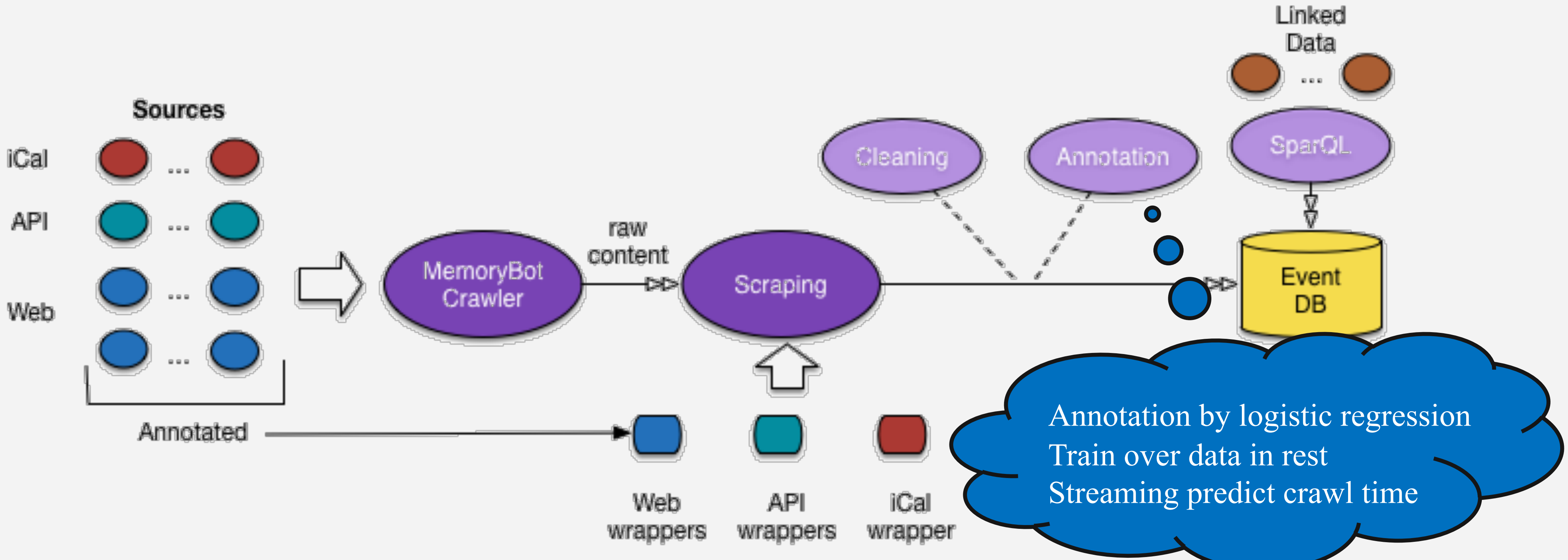
Identify events that influence consumer behavior (product purchases, media consumption)

Events influence people

Before a football match, people buy beer, chips, …

Specific events influence specific people (requires user profiles)

A football fan does not play Angry Birds during a football match



Portugal Telecom use cases

MEO quadruple-play

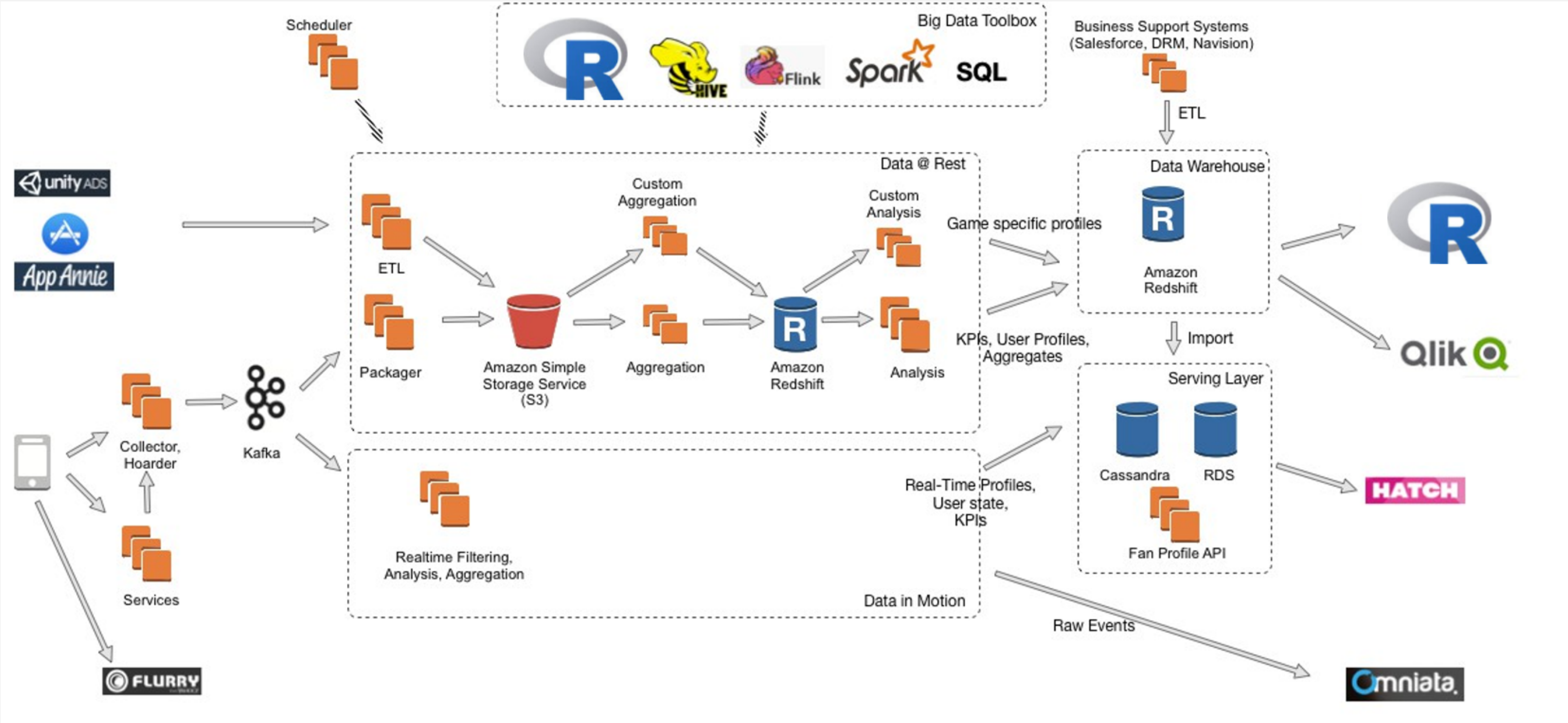
Features

- Internet
- TV (IPTV)
- Mobile phone
- Landline phone

Current challenges

- Heterogeneous data
- Heterogeneous technical solutions
- Customers profiling
- Cross-domain recommendation
- 1TB/day

Rovio use cases



Development at Sztaki

iALS
- Flink already has explicit ALS
- The implementation of the implicit version is done
- Currently testing the algorithm's accuracy

Matrix factorization
- Distributed algorithm*
- We have a working prototype tested on smaller matrices but it still needs optimization

Logistic regression
- Implementation in progress
- It is based on stochastic gradient descent, but in Flink there is only a batch version
- Currently working on the gradient descent implementation

Metrics
- Implementation and testing is finished
- We need to create a pull request

*R. Gemulla et al, "Large scale Matrix Factorization with Distributed Stochastic Gradient Descent", *KDD 2011.*

Summary

- Scala is a great tool for building DSLs
- FlinkML's API is motivated by scikit-learn
- Streaming is a natural fit for ML predictors
- Online learning can outperform batch in certain cases
- The Streamline project builds on Flink, aims to contribute back as much of the results as possible