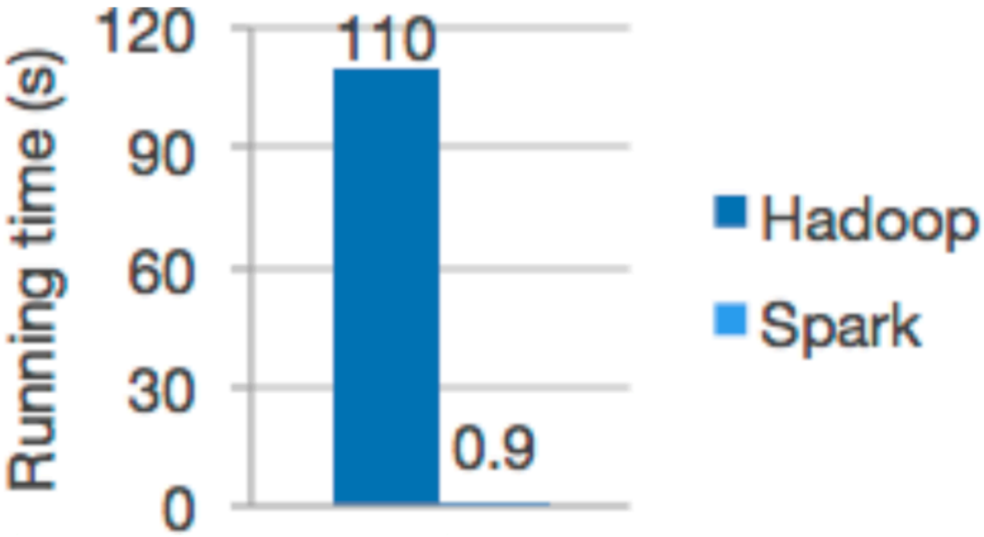
jbreak;

ML pipelines at OK

Dmitry Bugaychenko

@mail.ru group



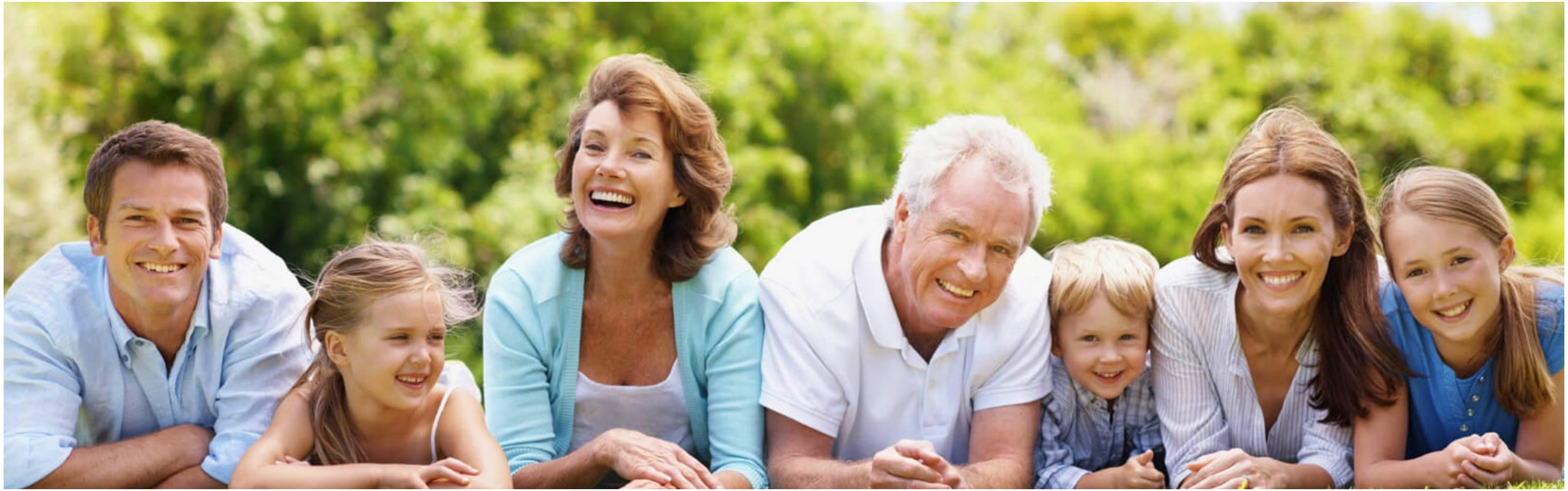
Logistic regression in Hadoop and Spark

OK is...

70 000 000+

monthly unique users

OK is…



800 000 000+ family links in the social graph

OK is…

A place where people share their positive feelings

OK is...

- 10000+ servers around the globe
- 1+Tb/s of outgoing traffic
- 400+ software components
- High Load, Big Data, Fault Tolerance...

OK is...

- 3 Hadoop clusters
- 30+ petabytes storage (+20Tb daily)
- 10000+ cores
- 40+ TB RAM
- 300+ regular jobs

Data Mining in 3 Steps

- Get the data

Data Mining in 3 Steps

- Get the data
- Find unobvious dependencies

Data Mining in 3 Steps

- Get the data
- Find unobvious dependencies
- Use them to make decisions

Data Mining with Machine Learning

- Get the data
- Find unobvious dependencies
  - By *fitting* parameters of a mathematical model
- Use them to make decisions

ML Tasks at OK

- Find the good stuff
  - Recommender systems
- Find the bad stuff
  - Anti-spam systems
- Understand the users
  - Explorative analytics

Smart Data Toolbox at OK

scikit learn

TEZ

HIVE

Spark

kafka

TensorFlow

hadoop

dmlc XGBoost

Apache Zeppelin

+ableau

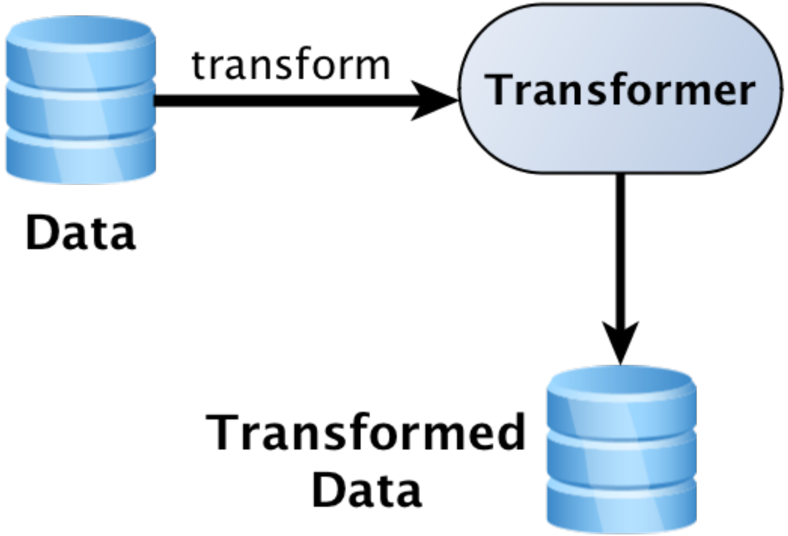
samza

Parquet

Spark ML Pipelines

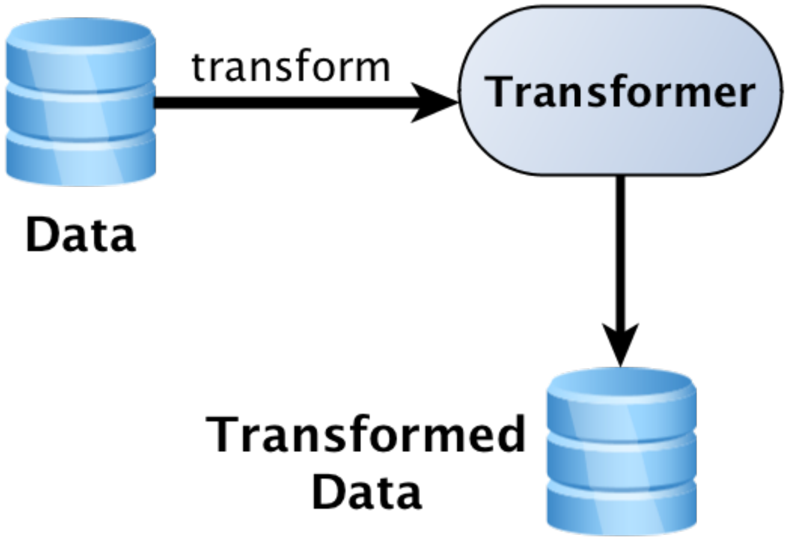
Spark ML Pipelines

Transformer

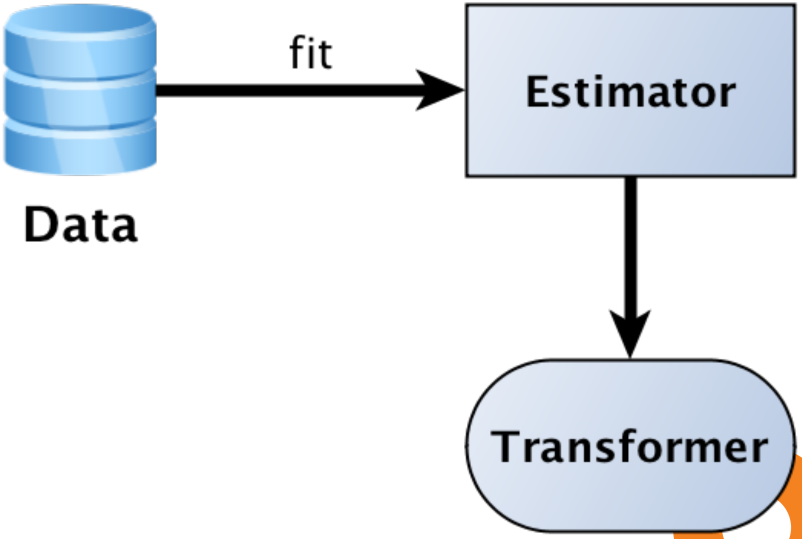


Spark ML Pipelines

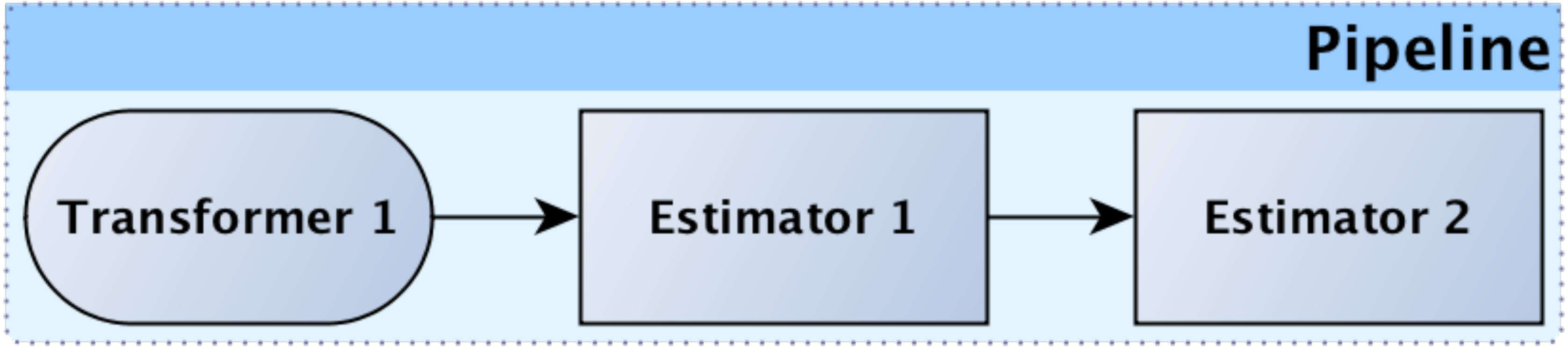
Transformer



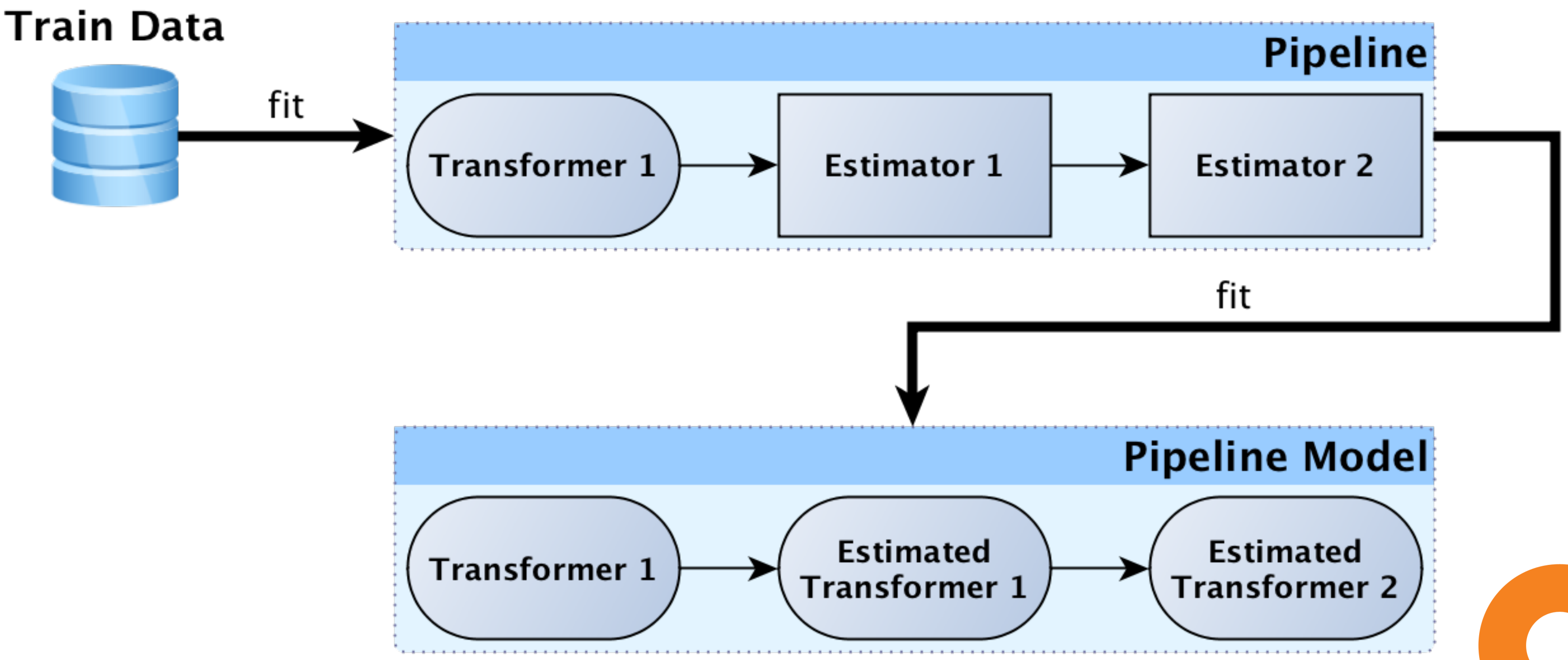
Estimator



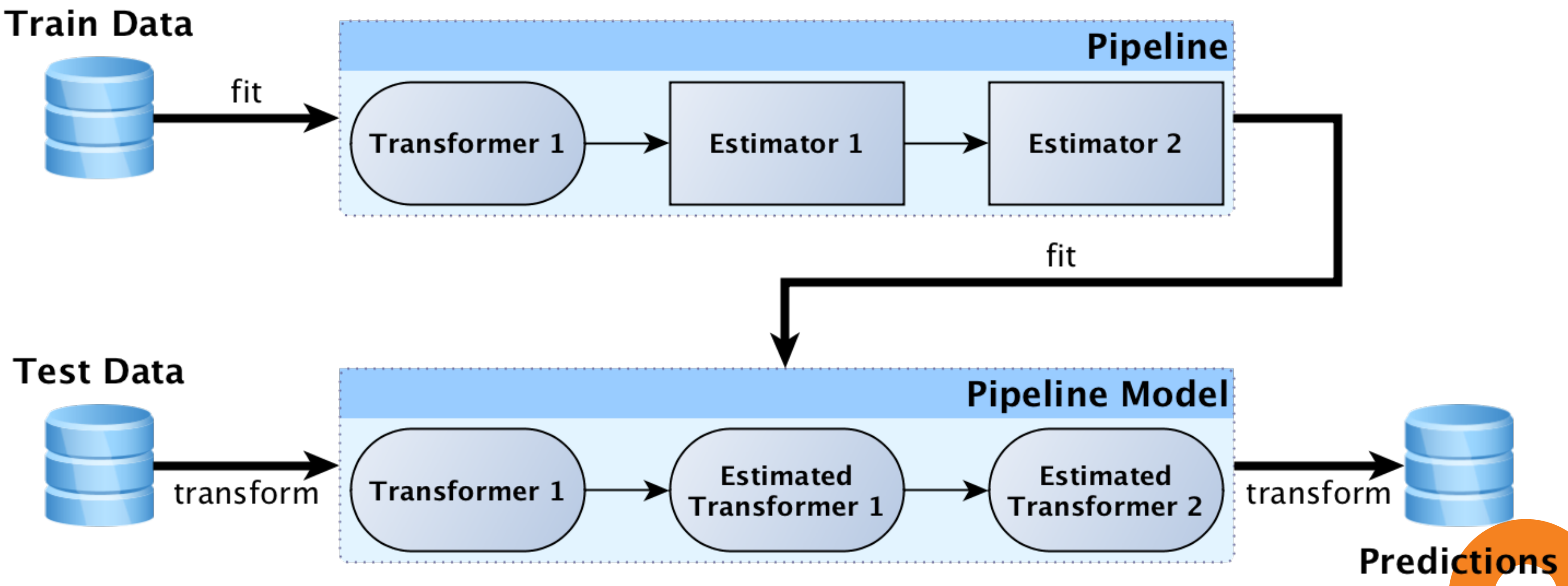
Spark ML Pipelines



Spark ML Pipelines



Spark ML Pipelines



Meet the OK ML Pipelines

- Open source project: https://github.com/odnoklassniki/ok-ml-pipelines
- Extends Spark ML pipelines for
  - Flexible dataflow organization
  - Better cluster utilization
  - ML scaling and operationalizing

Example task

- Churn prediction
- Input: 10M users with 200+ attributes
- Output: user’s retention
- Method:
  - Find unobvious dependencies
    - Build a prediction model
    - Analyze the models weights
  - Use them to make decisions
  - Profit!

The first pipeline: Extract features

```
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
      .withColumns("label", "numMessages", "numLikes")
      .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  new LogisticRegressionLBFSG()
))

val model = pipeline.fit(data)

val predictions = model.transform(data)
```

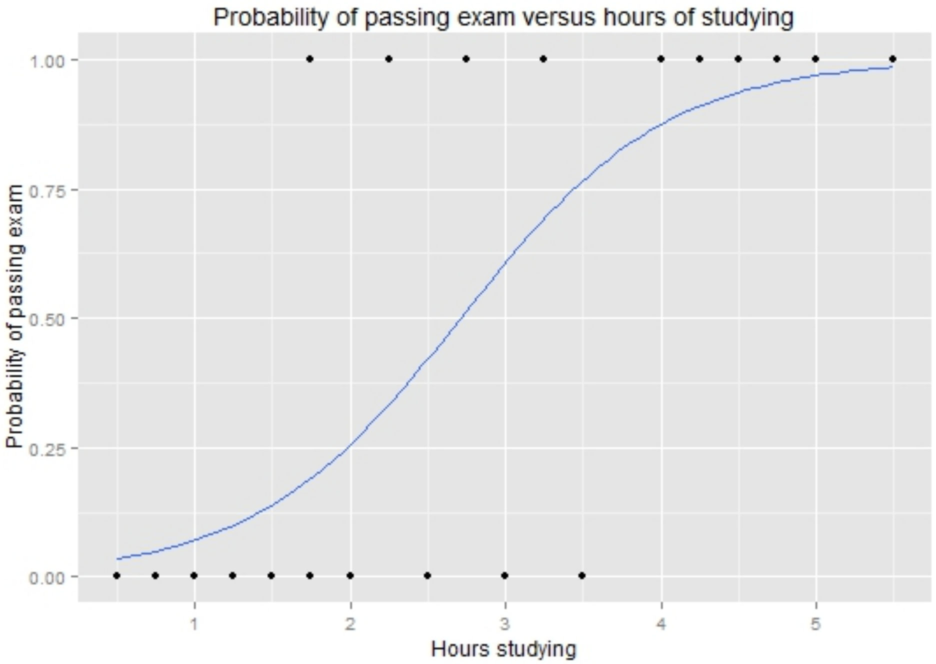
The first pipeline: Convert to vector

```
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
      .withColumns("label", "numMessages", "numLikes")
      .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  new LogisticRegressionLBFSG()
))

val model = pipeline.fit(data)

val predictions = model.transform(data)
```

The first pipeline: Choose the ML method

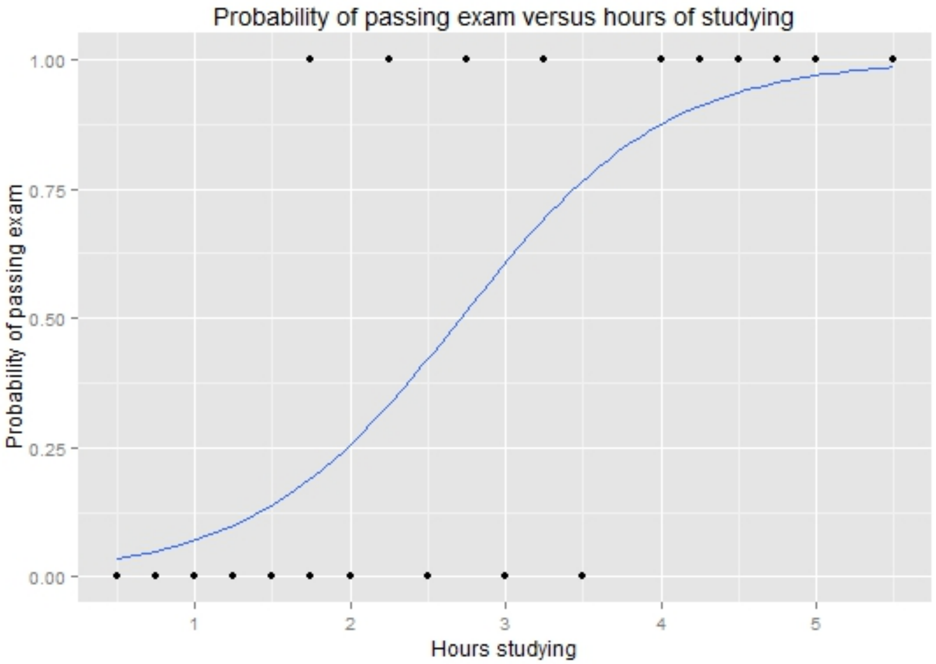
```
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
      .withColumns("label", "numMessages", "numLikes")
      .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  new LogisticRegressionLBFSG()
))

val model = pipeline.fit(data)

val predictions = model.transform(data)
```

Logistic Regression at a glance



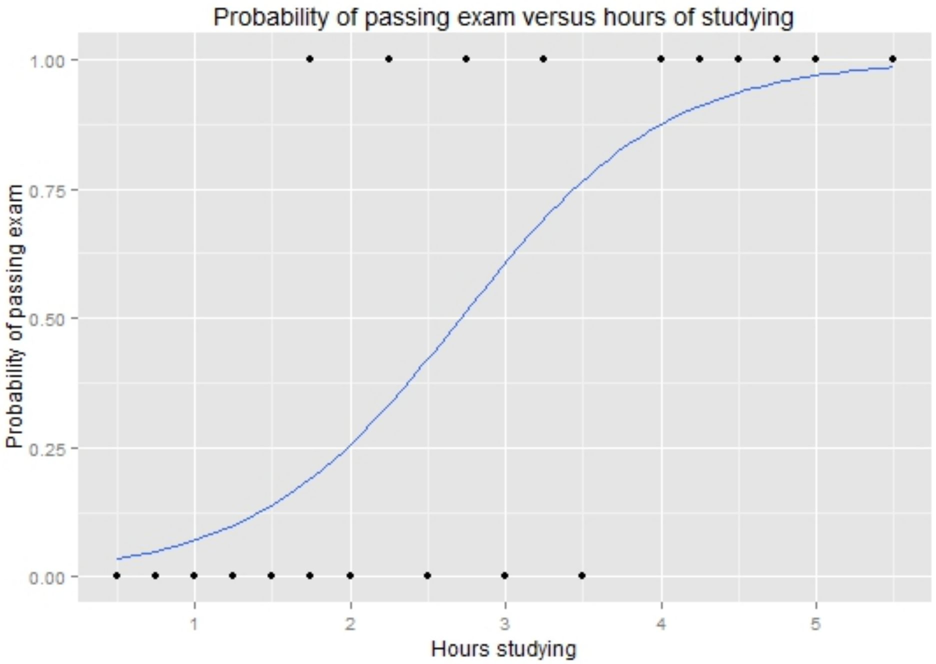
$$\frac{1}{1+e^{-(a \bullet x+b)}}$$

Logistic Regression at a glance



$$\frac{1}{1+\mathrm{e}^{-(a \bullet x+b)}}$$

Logistic Regression at a glance



$$\frac{1}{1+e^{-(a\bullet x+b)}}$$

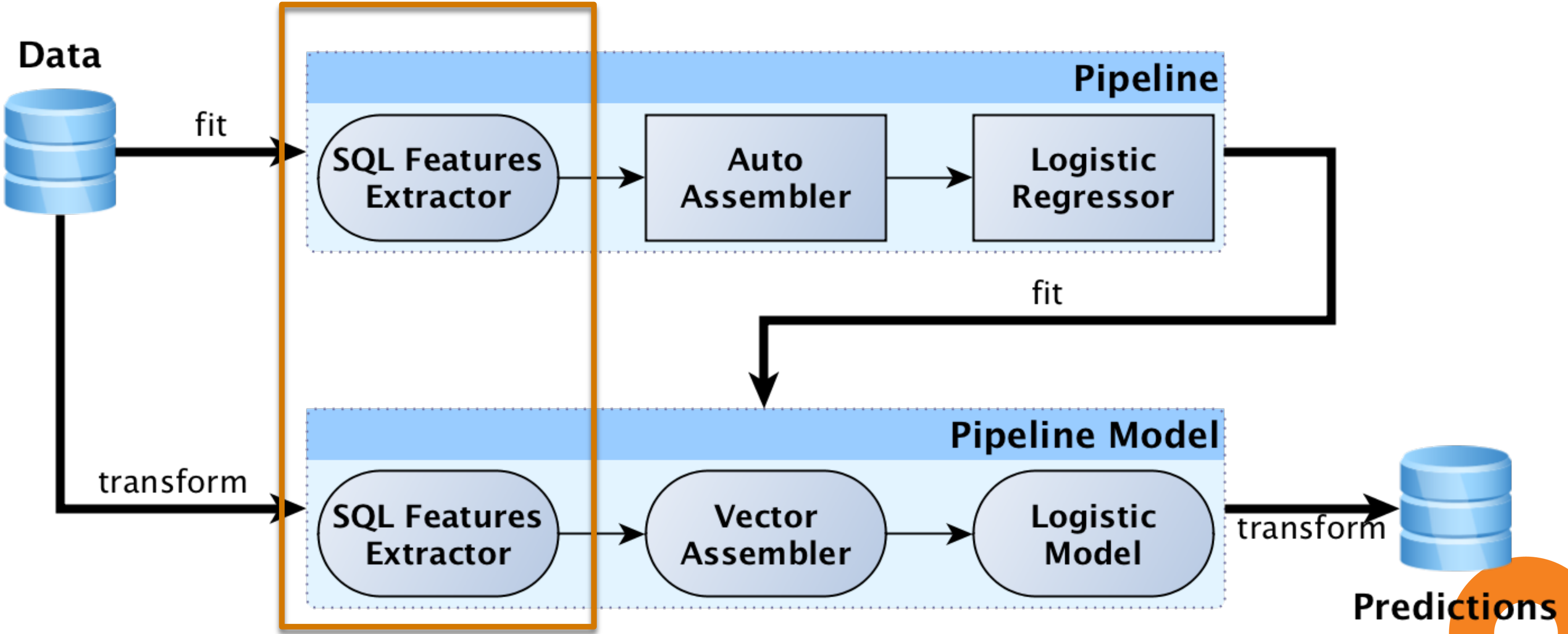
The first pipeline: Train the model

```
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
      .withColumns("label", "numMessages", "numLikes")
      .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  new LogisticRegressionLBFSG()
))

val model = pipeline.fit(data)

val predictions = model.transform(data)
```

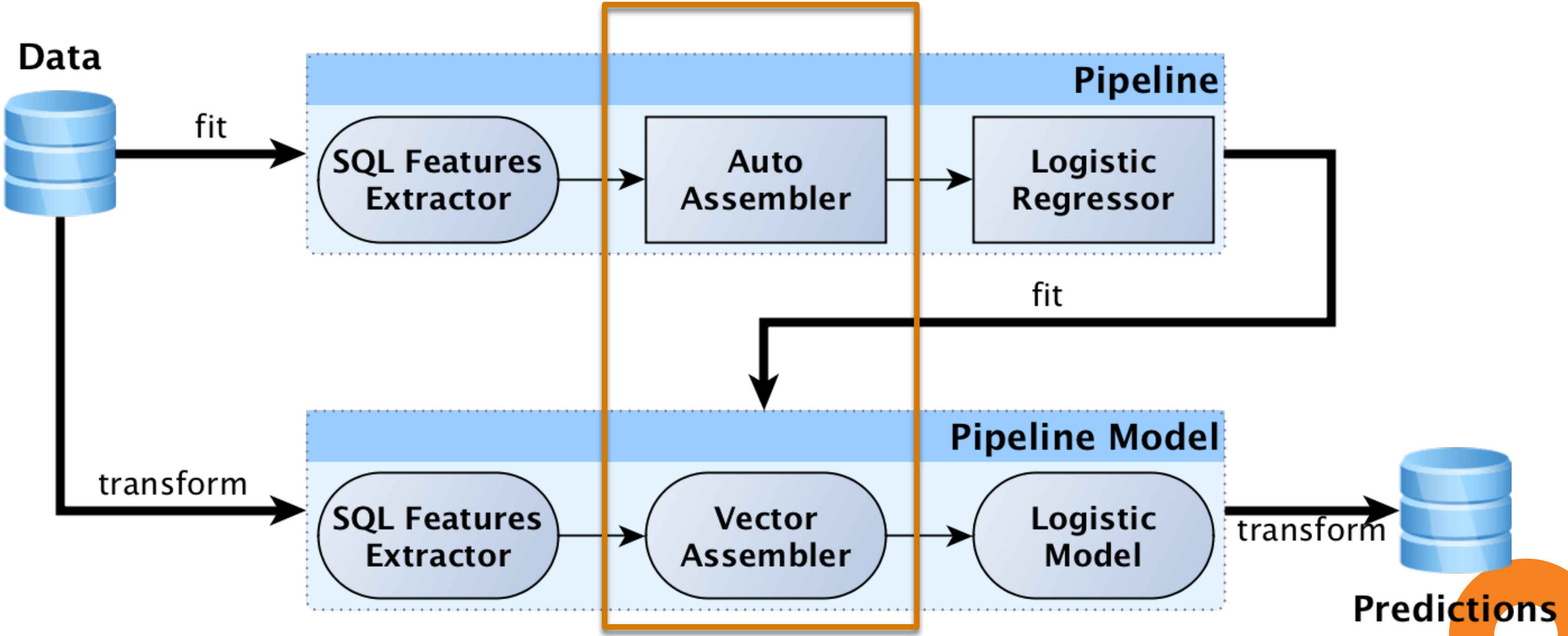
The first pipeline: Get the predictions

```scala
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
      .withColumns("label", "numMessages", "numLikes")
      .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  new LogisticRegressionLBFSG()
))

val model = pipeline.fit(data)

val predictions = model.transform(data)
```

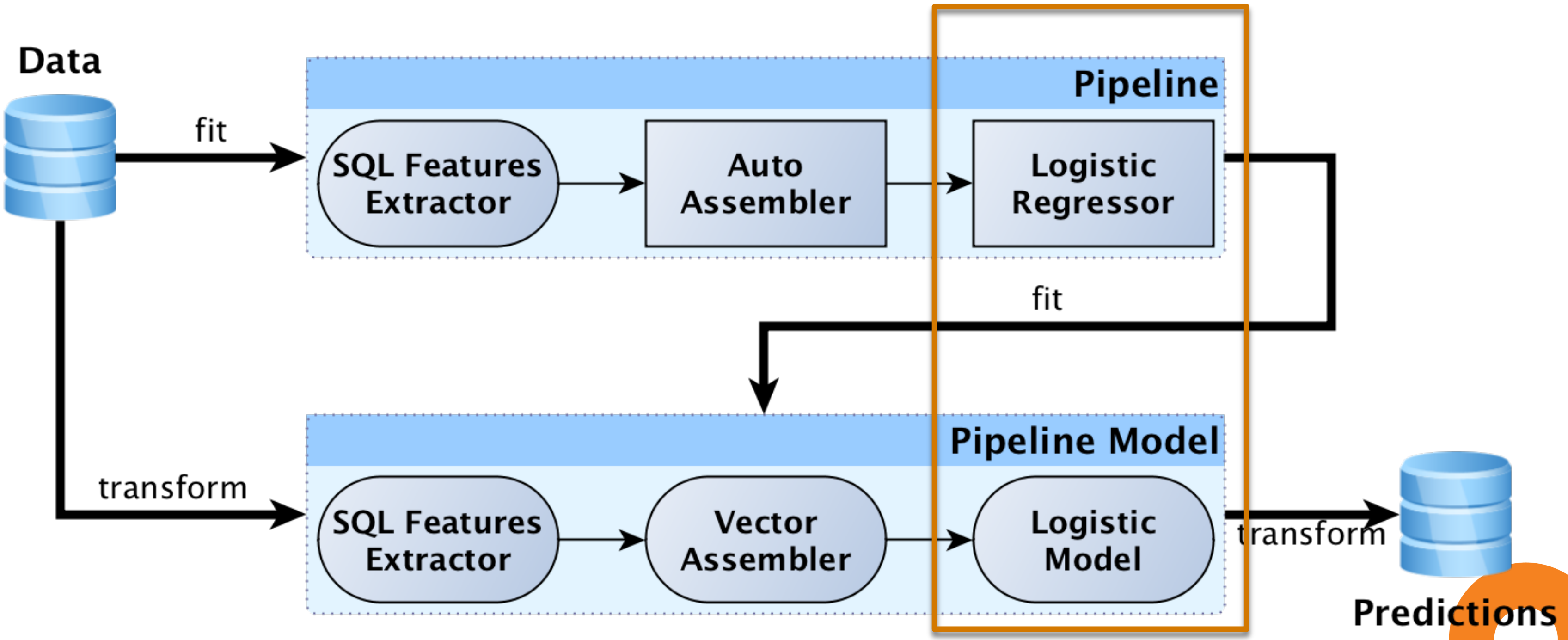
The first pipeline: Extract Features



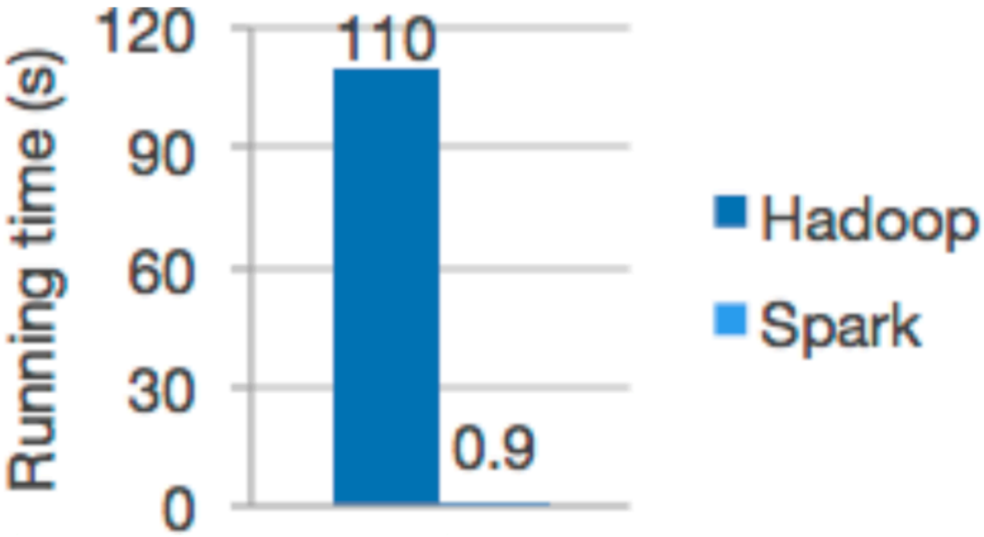
The first pipeline: Convert to vector



The first pipeline: Train the model

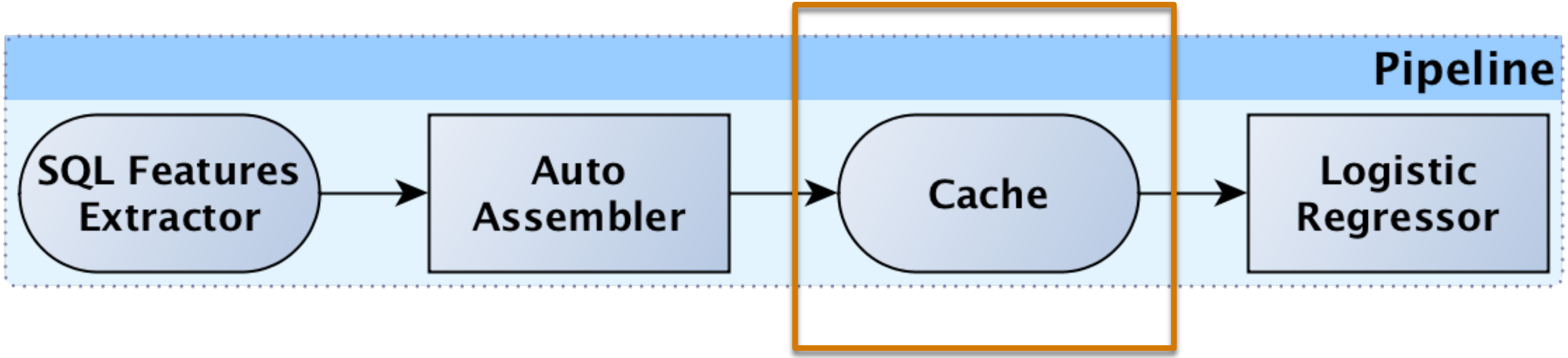


The most known histogram in ML world

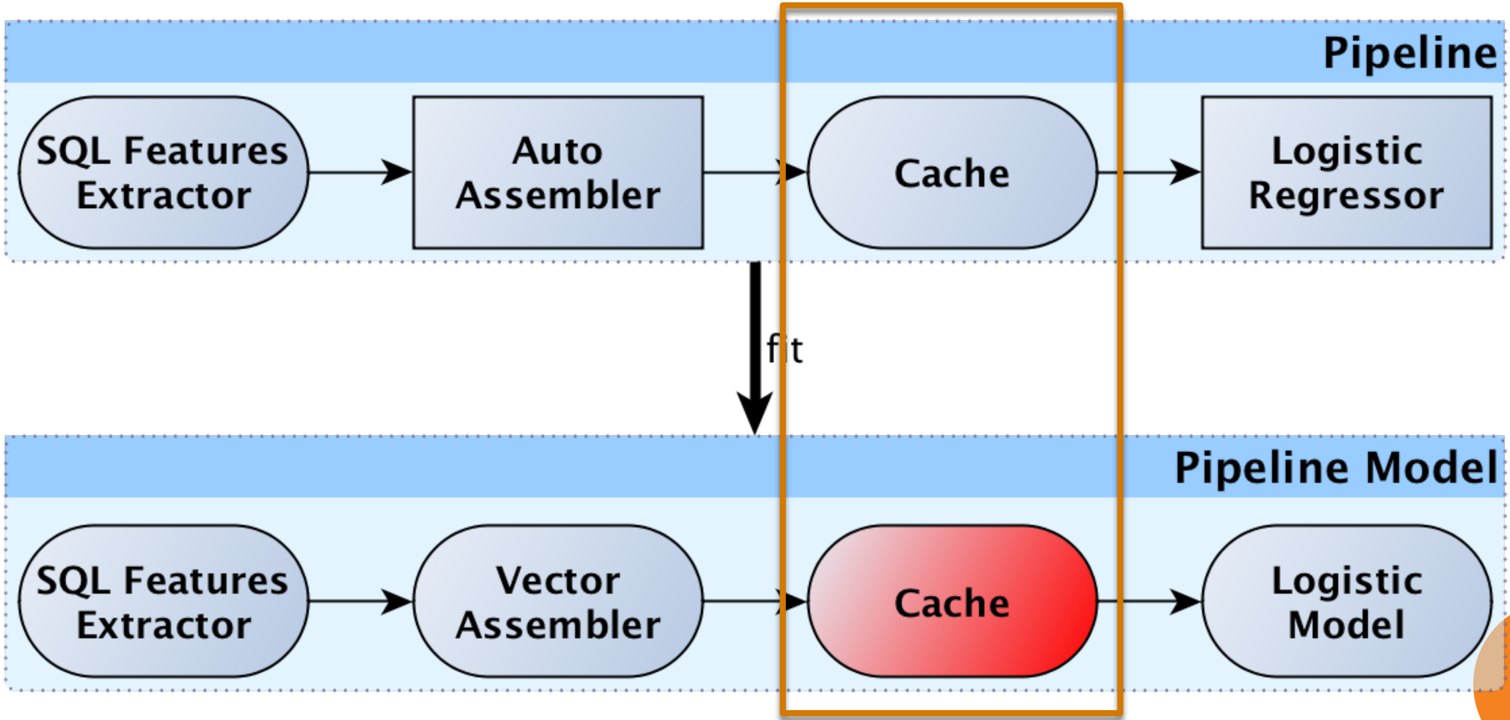


Logistic regression in Hadoop and Spark

Caching example: the simple way



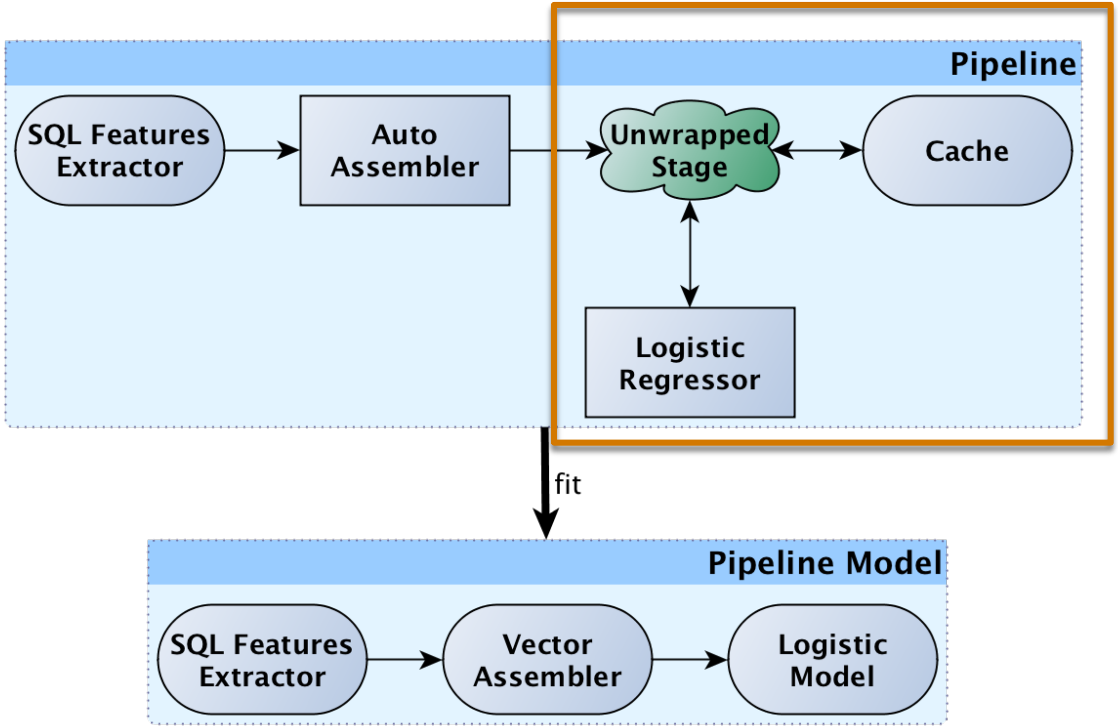
Caching example: the simple way failure



OK ML Pipelines: Unwrapped Stage

```scala
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
    .withColumns("label", "numMessages", "numLikes")
    .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  UnwrappedStage.cache(
    new LogisticRegressionLBFSG())
))
```

OK ML Pipelines: Unwrapped Stage



More cases of this kind

- Cache
- Repartitioning
- Random sampling
- Ordered cut
- Persist data to temp folder
- …

Back to the goal: exploring weights

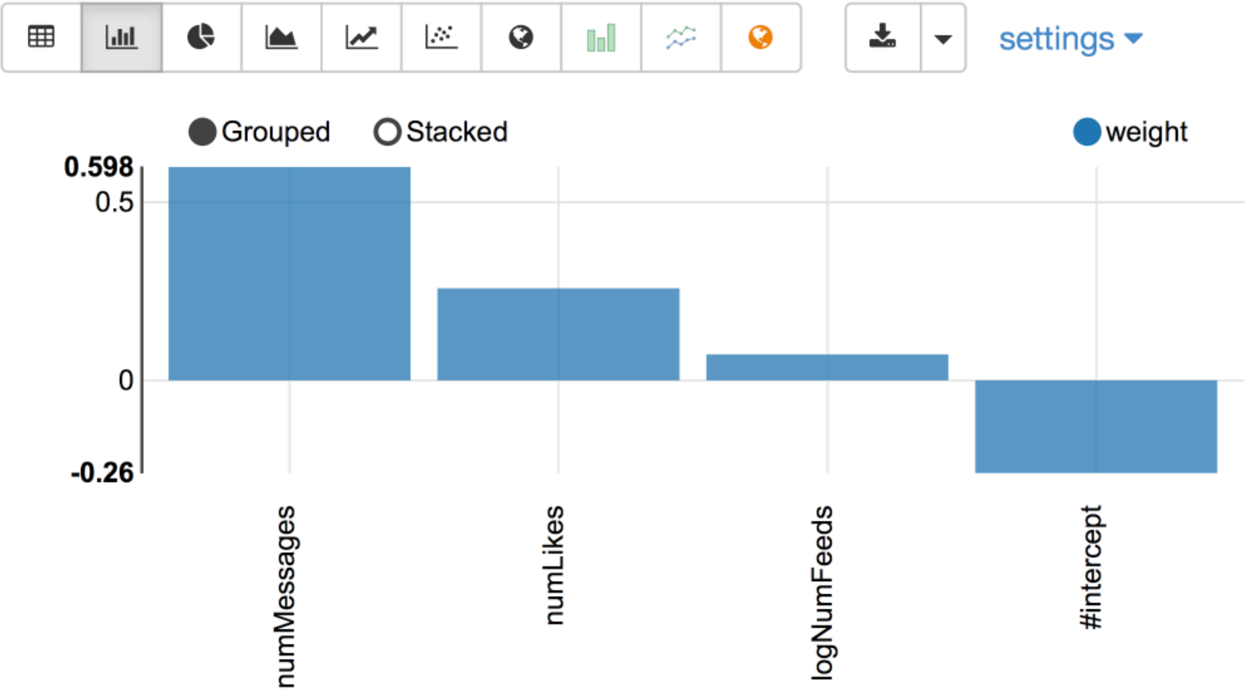
- We do not need the model itself
- We need some insights about the churn
- Lets look at the weights

Back to the goal: exploring weights

- We do not need the model itself
- We need some insights about the churn
- Lets look at the weights

```
model.write.save("/churn/simplestModel")

val weights = sqlc.read.parquet(
  "/churn/simplestModel/stages/*/weights/")

z.show(weights)
```

Back to the goal: exploring weights



But wait!

Spark ML does NOT work this way!

Back to the goal: exploring weights

- We do not need the model itself
- We need some insights about the churn
- Lets look at the weights

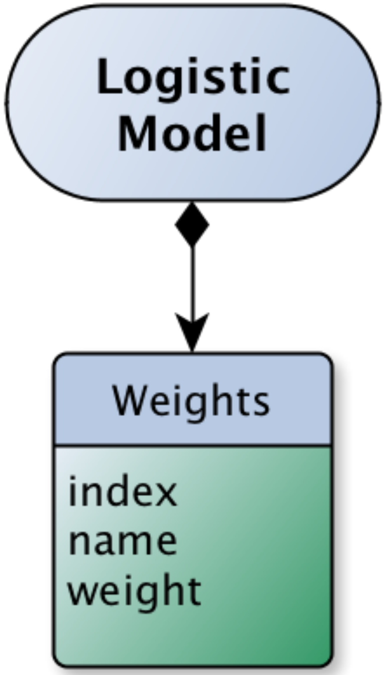
```
model.write.save("/churn/simplestModel")
```

```
val weights = sqlc.read.parquet(
  "/churn/simplestModel/stages/*/weights/")
```

```
z.show(weights)
```

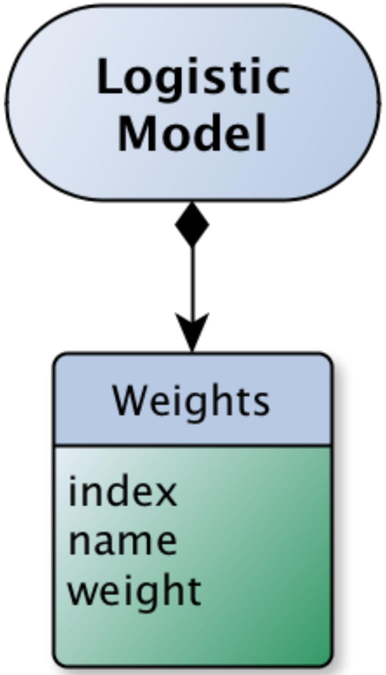
OK ML Pipelines: Model Summary

- Collection of named DataFrame's
- OK ML estimators produces summary
- Saved as parquet files



OK ML Pipelines: Model Summary

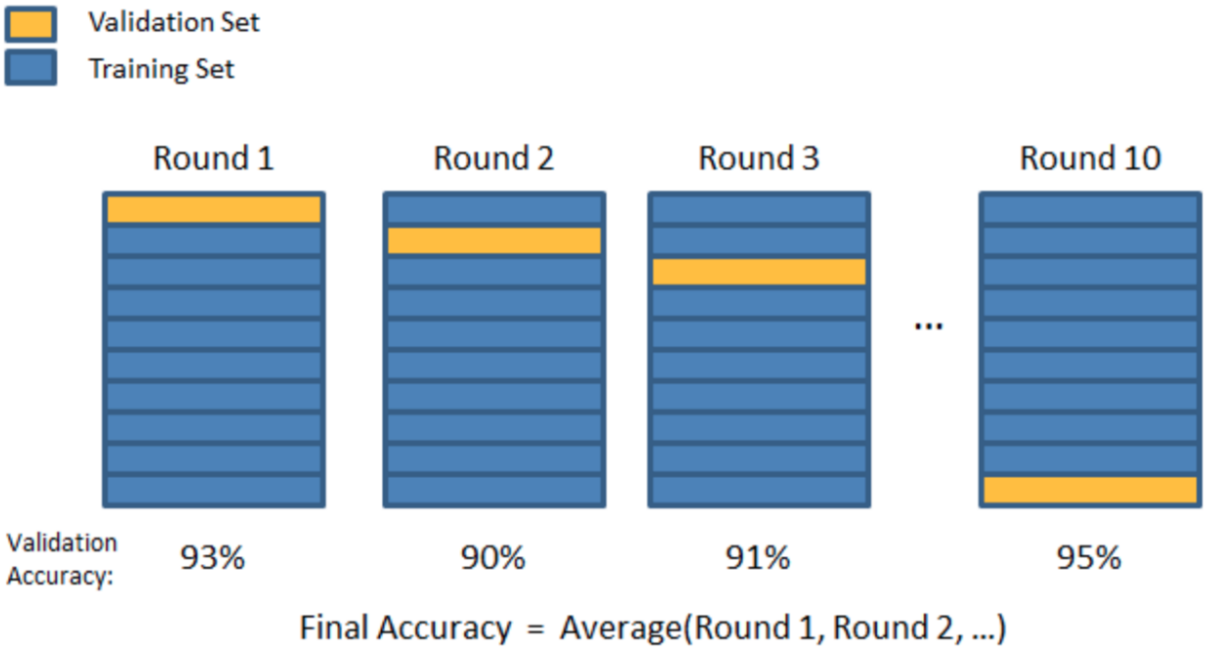
- Collection of named DataFrame’s
- OK ML estimators produces summary
- Saved as parquet files
- Spark ML estimator might be wrapped to produce summary



Back to the goal: how good is the model?

- Does the model make sense?
- We need to cross-validate the quality

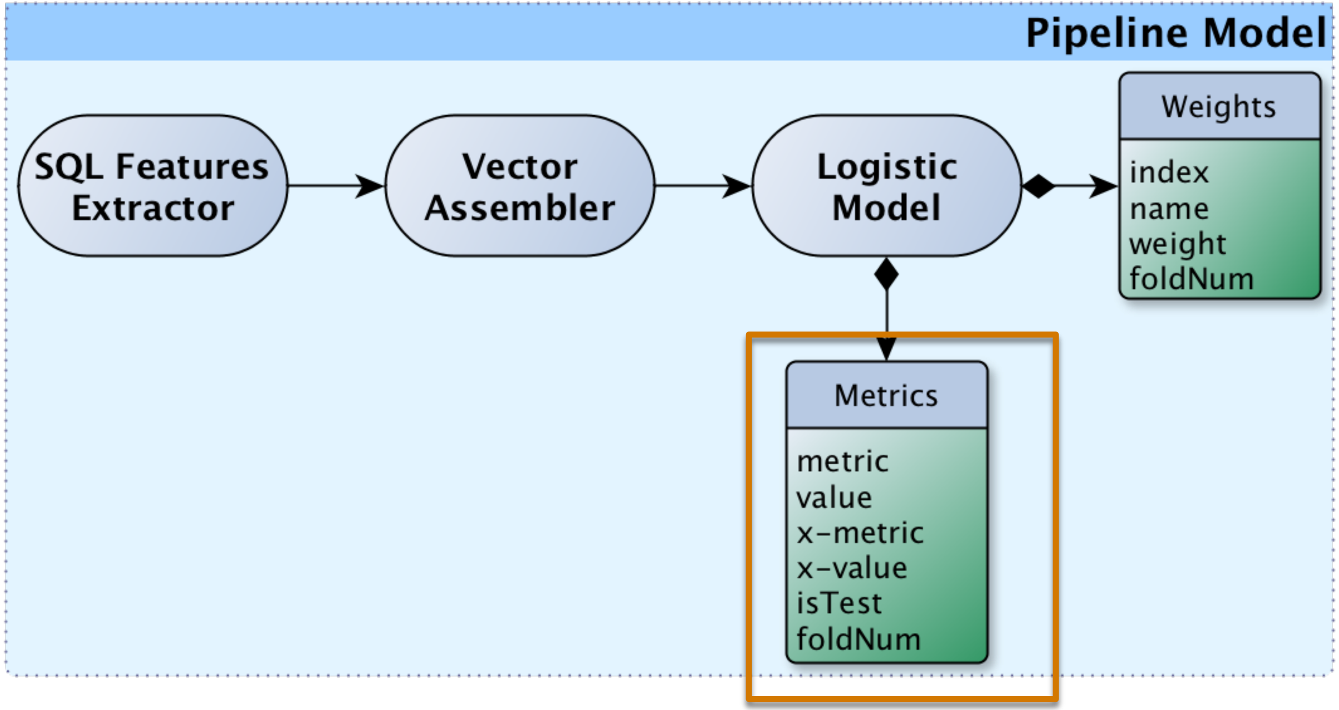
Cross-validation



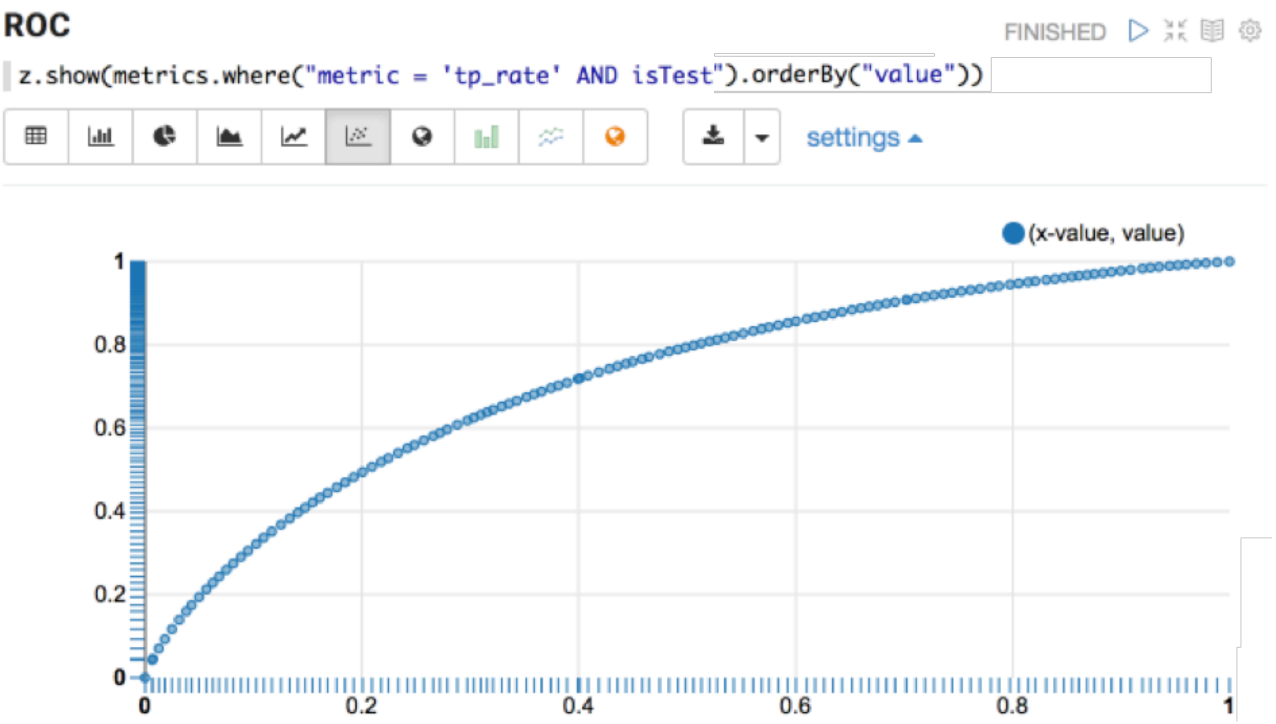
OK ML Pipelines: Evaluation

```
UnwrappedStage.cache(
  Evaluator.crossValidate(
    estimator = new LogisticRegressionLBFSG(),
    evaluator = new BinaryClassificationEvaluator())
```

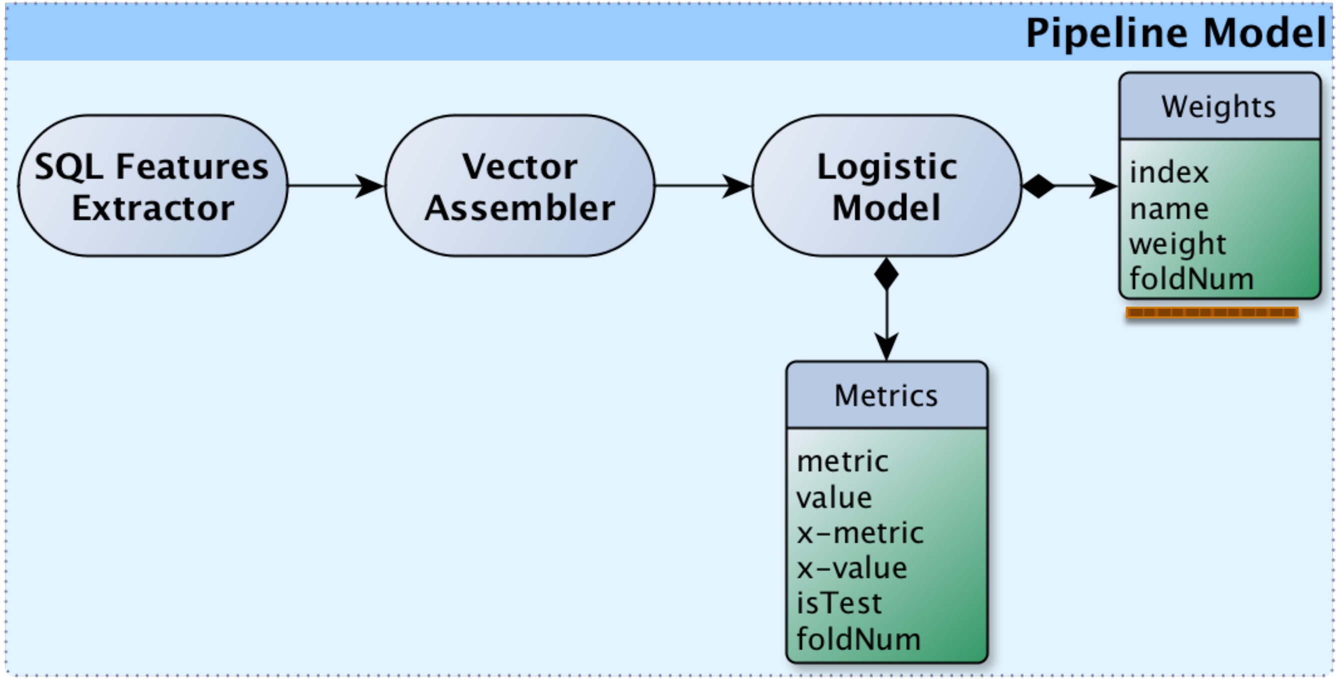
OK ML Pipelines: Evaluation



OK ML Pipelines: Evaluation



OK ML Pipelines: Evaluation



Evaluation: Spark ML vs. OK ML

Spark ML

- Evaluator, producing a single scalar
- Used for hyper parameters tuning

OK ML Pipelines

- Evaluator is a transformer converting predictions to metrics
- Injected using unwrapped stage
- Metrics block is added to the model summary

OK ML Pipelines: Evaluation

- Binary classification evaluator
  - ROC, RP-curve, F1-plot
- Partitioned ranking evaluator
  - NDCG, AUC,…
- Post processing evaluator
  - collects stat for metrics
- Train/test evaluator
- Cross-validation

Some thoughts on cross-validation

- 10+1 models
- No shared mutable state
- 90% intersection on input

Some thoughts on cross-validation

- 10+1 models
- No shared mutable state
- 90% intersection on input
- Single trainer utilizes less than 100 cores
- 5000+ cores in the cluster…

Spark ML CrossValidator

```
val splits = MLUtils.kFold(dataset.toDF.rdd, $(numFolds), $(seed))
splits.zipWithIndex.foreach { case ((training, validation), splitIndex) =>
  val trainingDataset = sparkSession.createDataFrame(training, schema).cache()
  val validationDataset = sparkSession.createDataFrame(validation, schema).cache()
```

- Sequential
- Cache each fold separately

Spark ML CrossValidator

```scala
val splits = MLUtils.kFold(dataset.toDF.rdd, $(numFolds), $(seed))
splits.zipWithIndex.foreach { case ((training, validation), splitIndex) =>
  val trainingDataset = sparkSession.createDataFrame(training, schema).cache()
  val validationDataset = sparkSession.createDataFrame(validation, schema).cache()
```

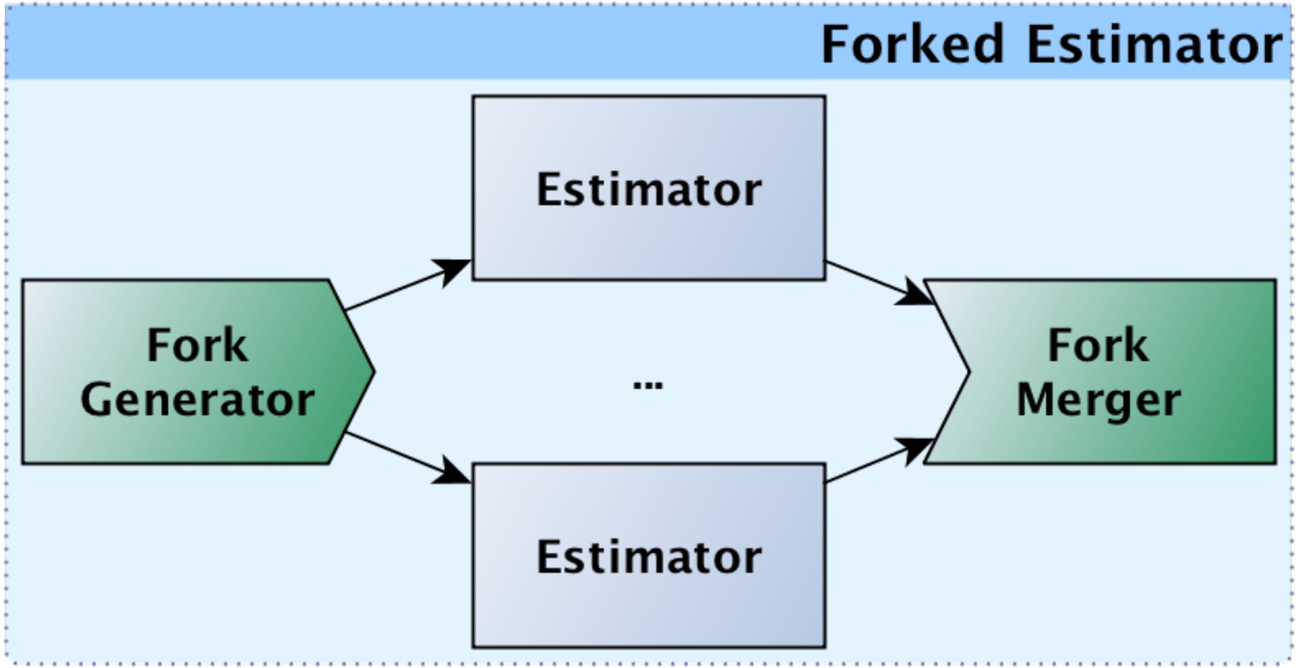
- Sequential
- Cache each fold separately

OK ML CrossValidator

```
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
    .withColumns("label", "numMessages", "numLikes")
    .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  UnwrappedStage.cache(
    Evaluator.crossValidate(
      estimator = new LogisticRegressionLBFSG(),
      evaluator = new BinaryClassificationEvaluator(),
      parallel = true)
)))
```

- Cache shared source data
- Train folds in parallel

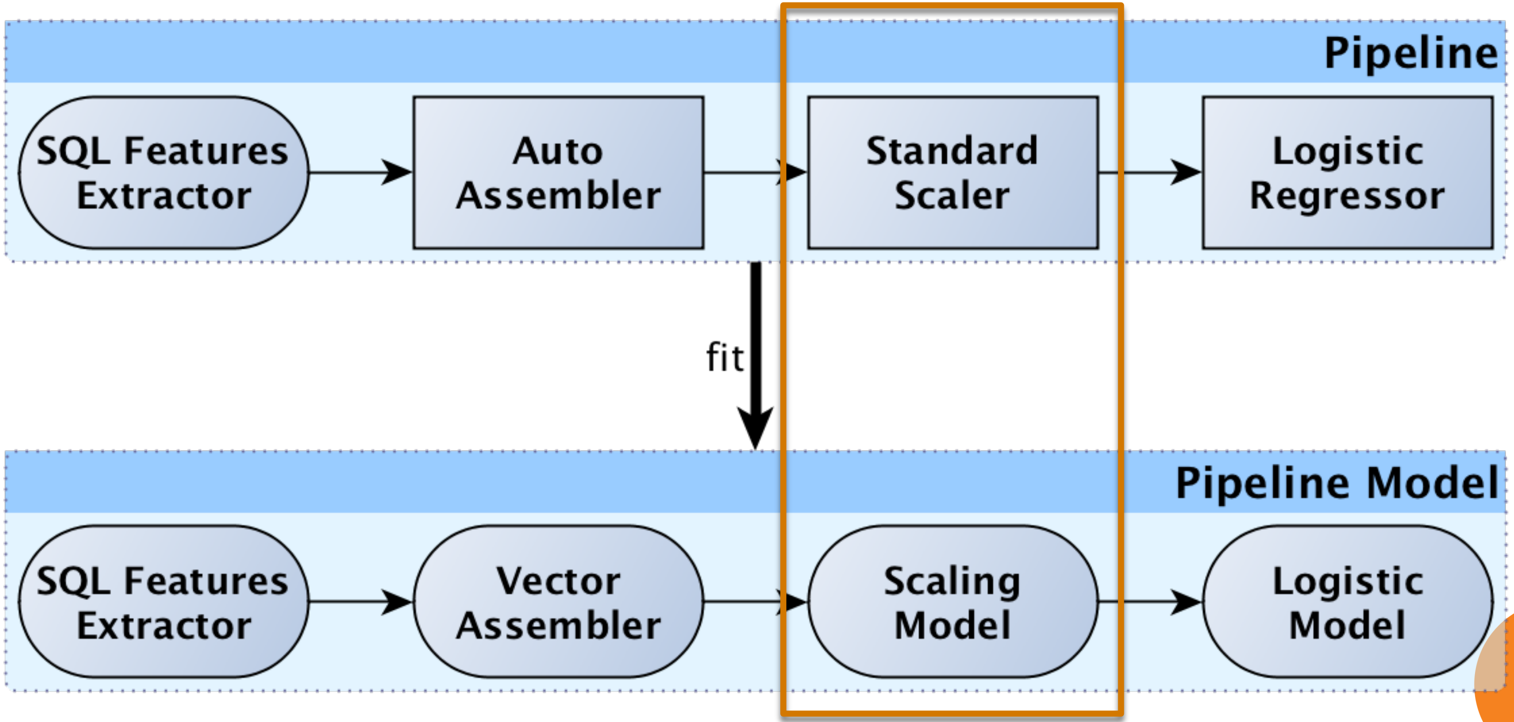
OK ML Pipelines: Forked Estimator



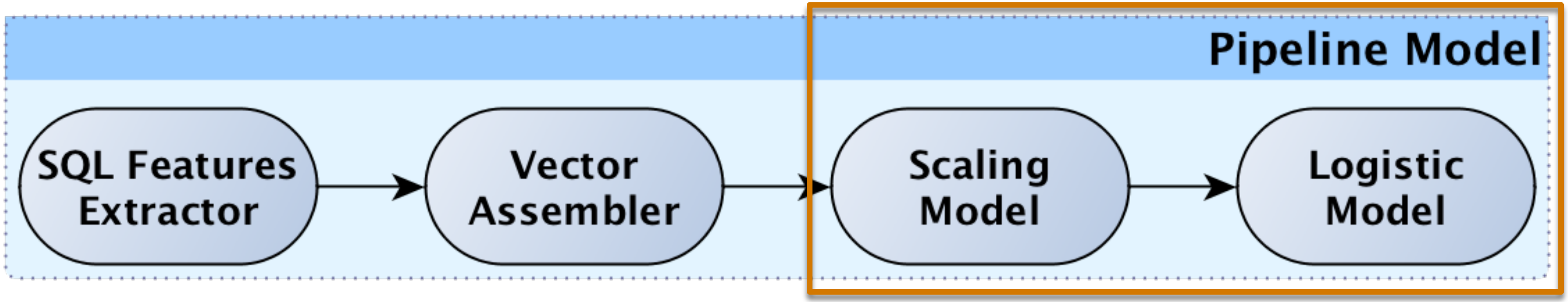
OK ML Pipelines: Forked Estimator

- Model segmentation
- Multi-class classification
- Feature selection
- Cross validation
- Grid search
  - work in progress

Cherry on the pie: Features scaling

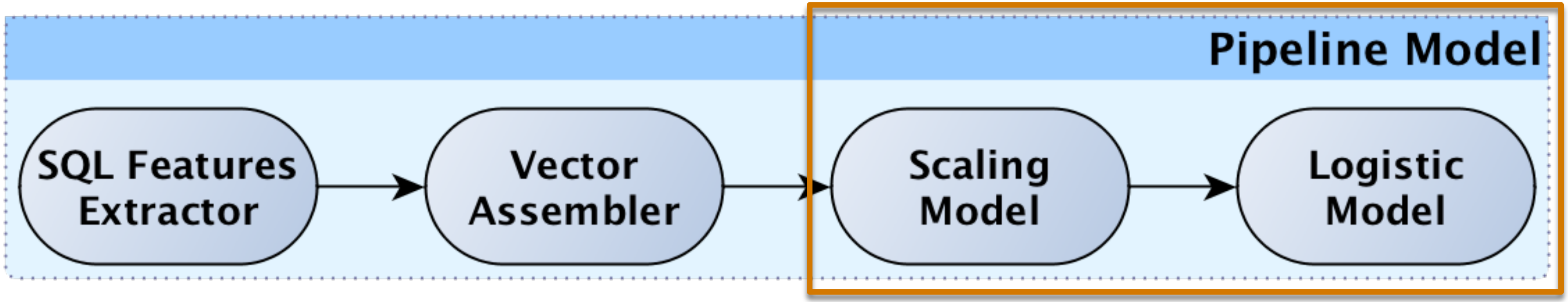


Spark ML Pipelines: Scaling



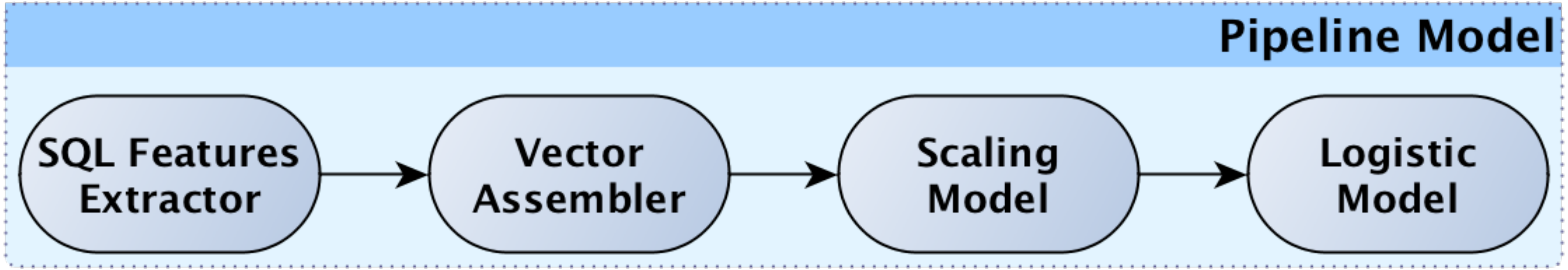
- Two models to move to prod
- Two set of weights to keep

Spark ML Pipelines: Scaling



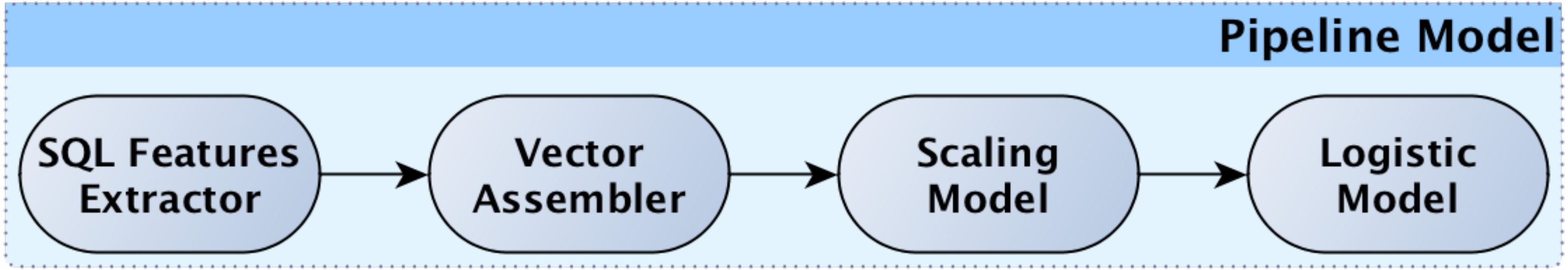
- Two models to move to prod
- Two set of weights to keep

Spark ML Pipelines: Scaling



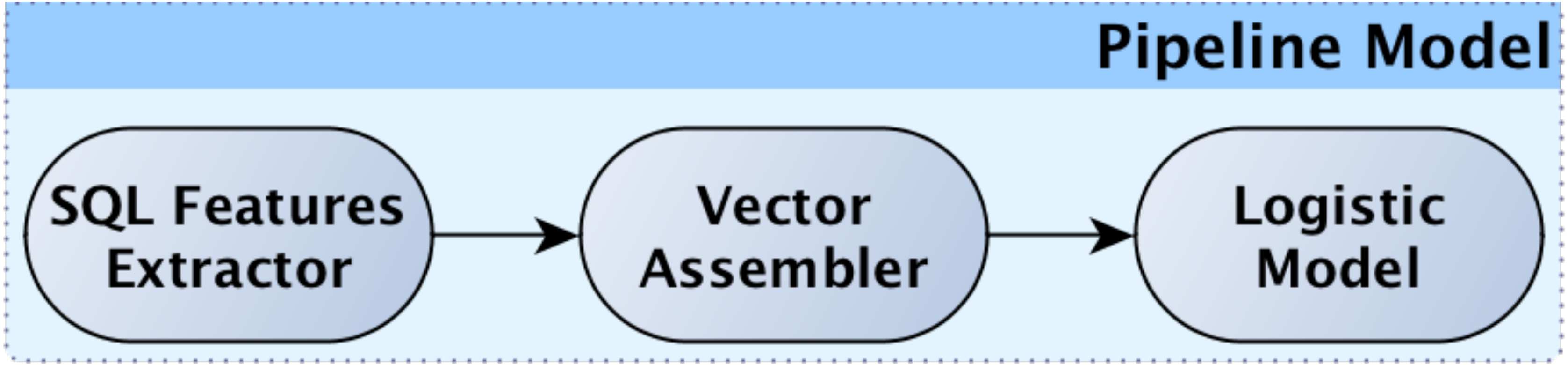
$$\frac{x_i - \bar{x}_i}{sd(x_i)}$$

Spark ML Pipelines: Scaling



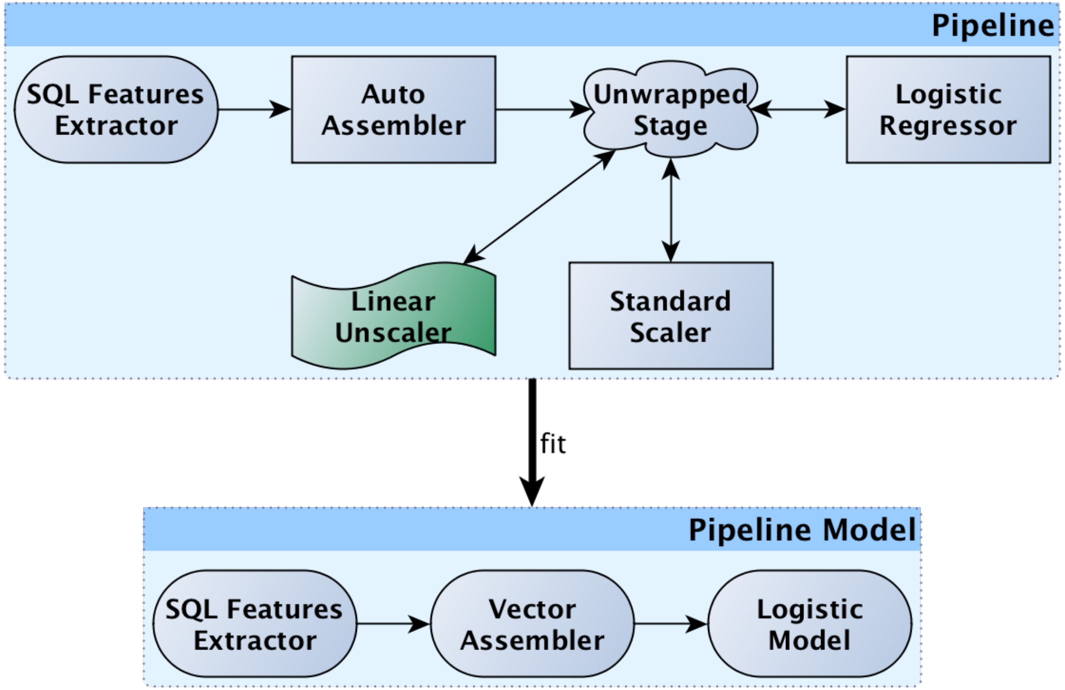
$$\frac{x_i - \overline{x}_i}{sd(x_i)} \qquad \frac{1}{1 + e^{-(a \bullet x + b)}}$$

OK ML Pipelines: inline scaling

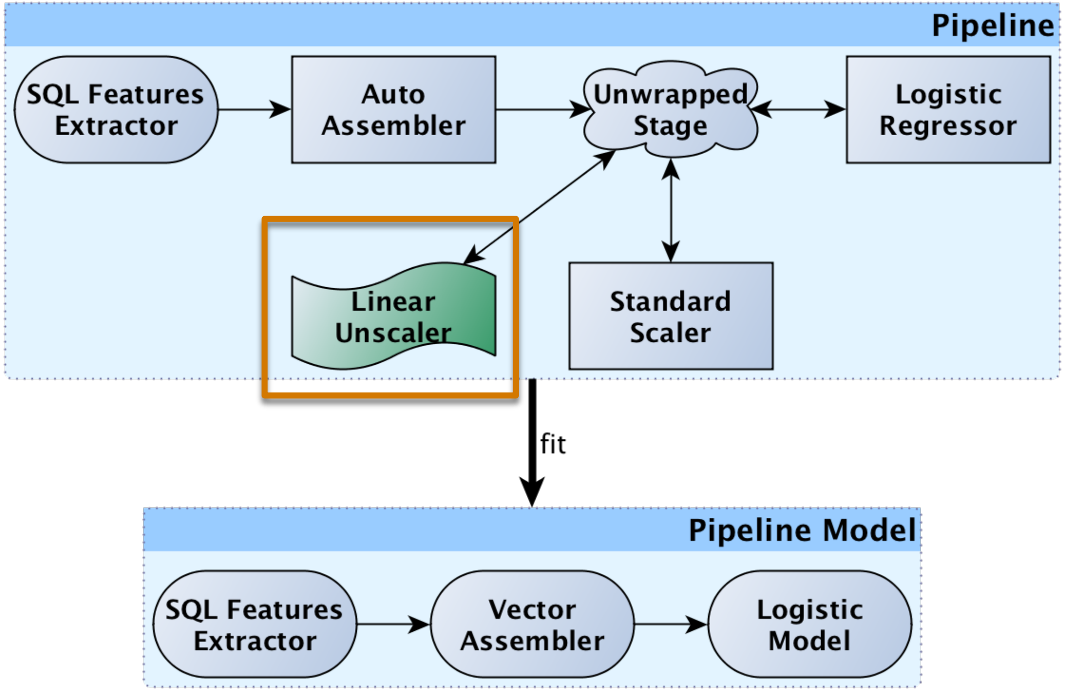


$$a'_i = \frac{a_i}{sd(x_i)}; b' = b - a' \bullet \overline{x}$$

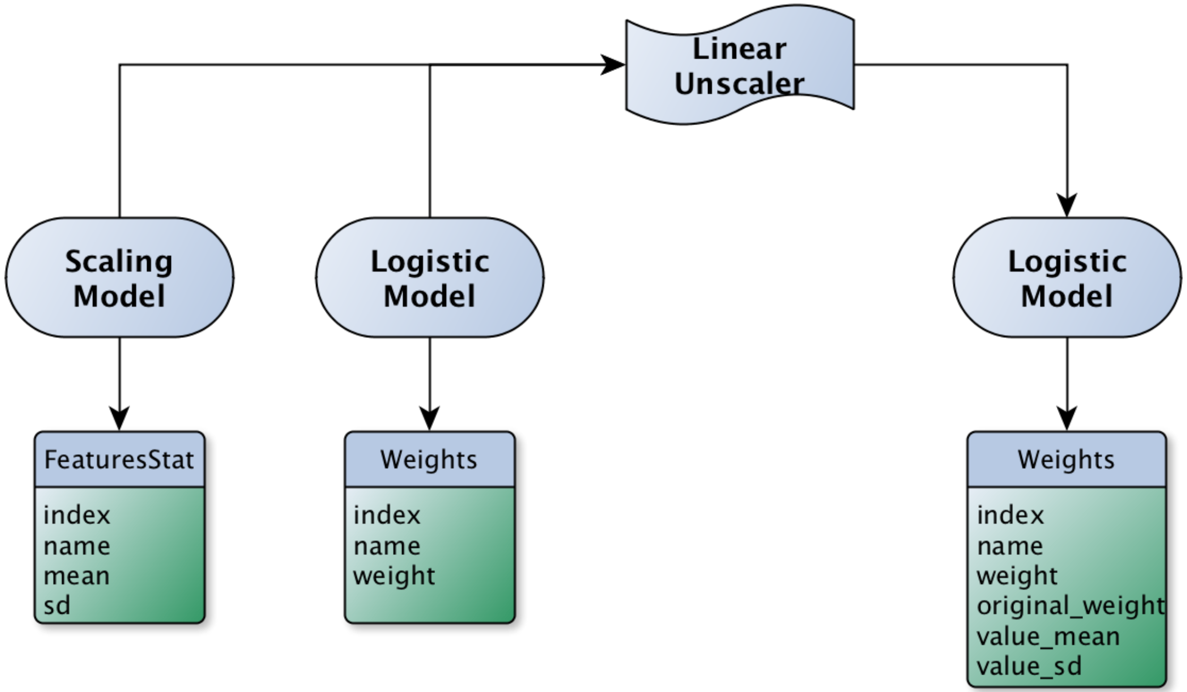
OK ML Pipelines: Model Transformer



OK ML Pipelines: Model Transformer



OK ML Pipelines: Model Transformer



OK ML Pipelines: inline scaling

```
val pipeline = new Pipeline().setStages(Array(
  new ColumnsExtractor()
    .withColumns("label", "numMessages", "numLikes")
    .withExpresions("logNumFeeds" -> "LOG(numFeeds + 1)"),
  new AutoAssembler().setColumnsToExclude("label"),
  Scaler.scale(UnwrappedStage.cache(
    Evaluator.crossValidate(
      estimator = new LogisticRegressionLBFSG(),
      evaluator = new BinaryClassificationEvaluator(),
      parallel = true)
))))
```

Even more cool stuff

- NLP transformers:
  - Language detection
  - Language aware analyzer
  - URL eliminator
  - Vectorizer
  - N-gram extractor
  - Trending term detection
  - LDA
    - work in progress
- Stat utils
  - Vector stat collector
  - Extended online summarizer
- Learning algorithms
  - Matrix LBFGS
  - DSRVGD
  - CRR

How to use it

Spark 2.2 (sbt):

```
libraryDependencies +=
  "ru.odnoklassniki" %%
  "ok-ml-pipelines" %
  "0.1-spark2.2" withSources()
```

Spark 1.6 (sbt):

```
libraryDependencies +=
  "ru.odnoklassniki" %%
  "ok-ml-pipelines" %
  "0.1-spark1.6" withSources()
```

Sources:

https://github.com/odnoklassniki/ok-ml-pipelines

Thank you for your attention!

Dmitry.Bugaychenko@corp.mail.ru