

# Перенос ML-алгоритма в распределенную среду

Дмитрий Бугайченко



**СБЕРБАНК**

*Всегда рядом*



# Немного о себе

Облачная платформа рекомендации **2019**

Автоматические А/В тесты **2018**

Рекомендации контента для витрин **2017**

Рекомендации друзей **2016**

Умная лента **2015**

Универсальная аналитическая платформа **2014**

Рекомендации Групп **2013**

Рекомендации музыки **2012**

Работа в Mail.ru **2011**

Преподаватель СПбГУ **2009**

Кандидат физ-мат наук **2008**

**10 лет  
в DM**

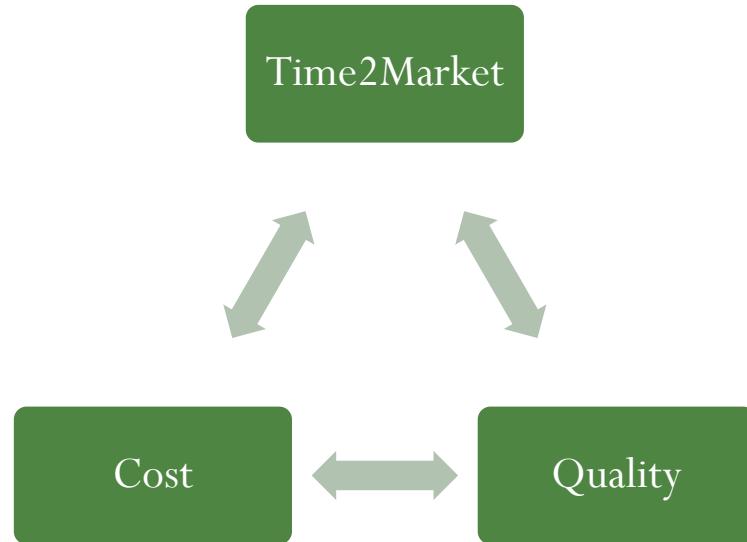
**18 лет  
в IT**



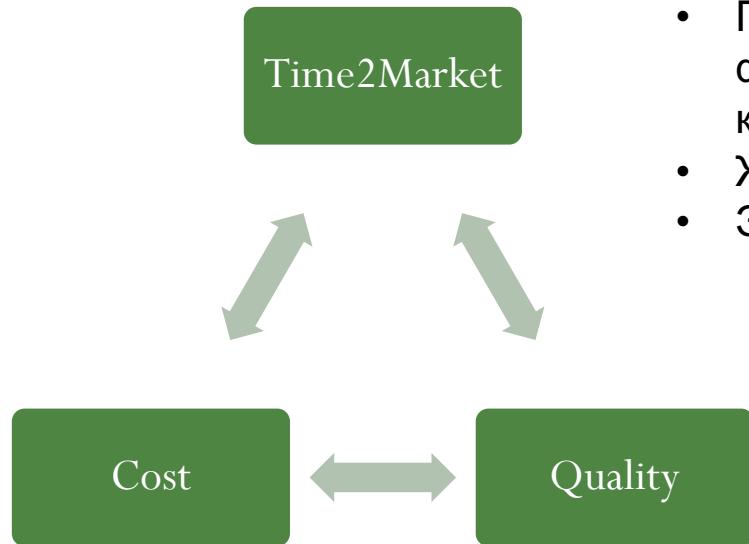
# Экономика

---

# Вечная дилемма



# Вечная дилемма



Массивно-параллельный  
подход:

- Популярные МЛ-  
фреймворки работают из  
коробки
- Железо влетает в копеечку
- Эксплуатация еще больше

# Вечная дилемма

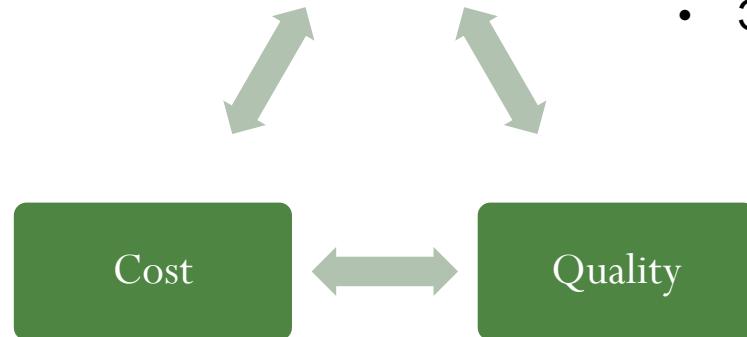
Распределенный подход:

- Стоковое дешевое железо
- Простая эксплуатация
- Ограниченный набор моделей в стандартных фреймворках

Time2Market

Массивно-параллельный подход:

- Популярные МЛ-фреймворки работают из коробки
- Железо влетает в копеечку
- Эксплуатация еще больше



# Вечная дилемма

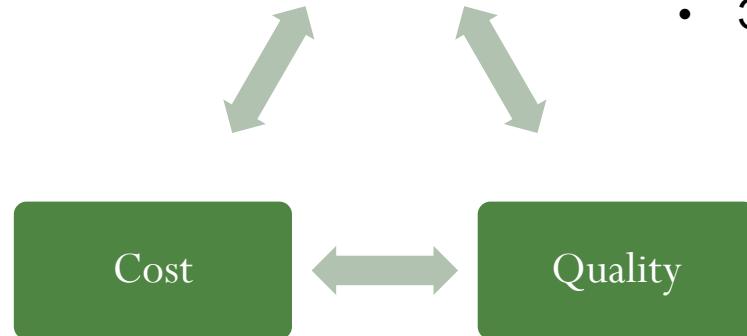
Распределенный подход:

- Стоковое дешевое железо
- Простая эксплуатация
- Ограниченный набор моделей в стандартных фреймворках

Time2Market

Массивно-параллельный подход:

- Популярные МЛ-фреймворки работают из коробки
- Железо влетает в копеечку
- Эксплуатация еще больше



Распределенный подход после курса MADE:

- Сначала делаем R&D по распределению алгоритма
- Тренируем модели быстро, дешево и качественно

# Правила распределенного МЛ

1. Не иди в распределенный МЛ пока финансы позволяют

# Правила распределенного МЛ

1. Не иди в распределенный МЛ пока финансы позволяют
2. Ищи источники параллелизма

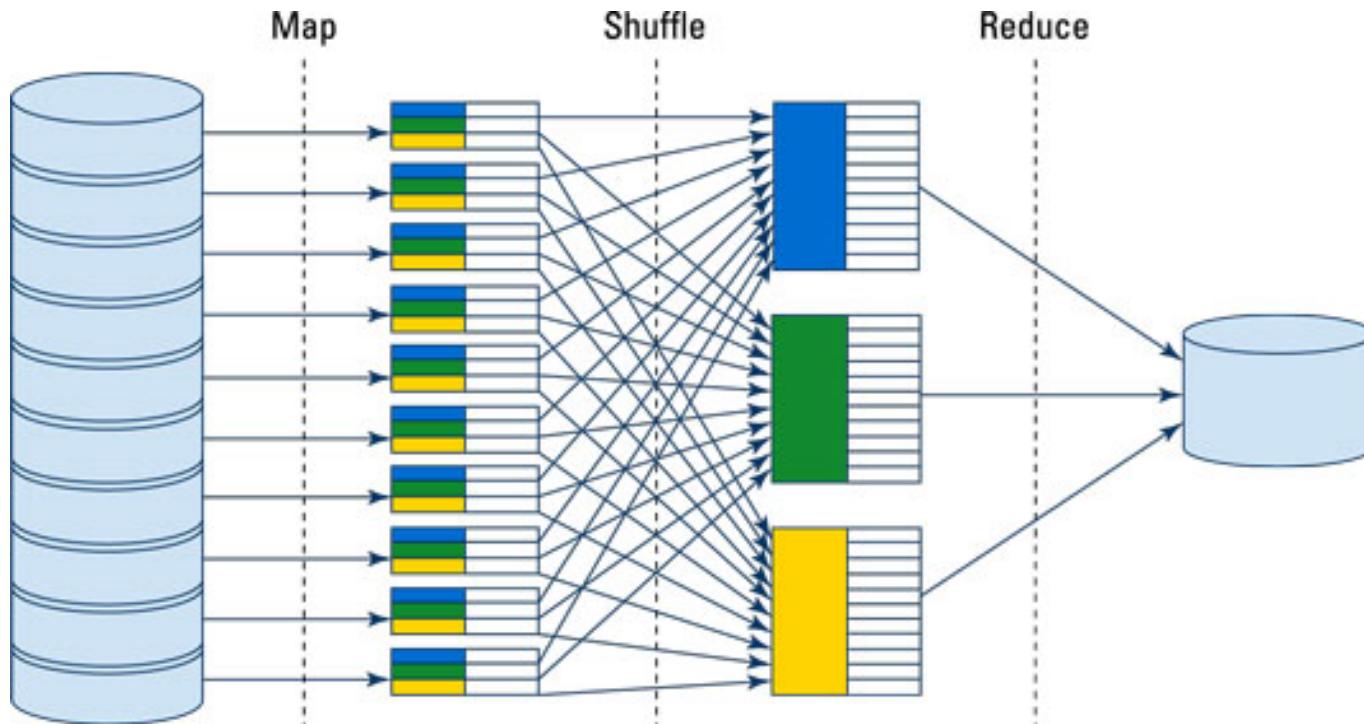
# Источники параллелизма в МЛ

- Параллелизм данных
  - Считаем части суммы градиента параллельно
- Параллелизм модели
  - Обновляем разные части модели параллельно
- Параллелизм задачи
  - Вычисляем модель для каждого «фолда» параллельно

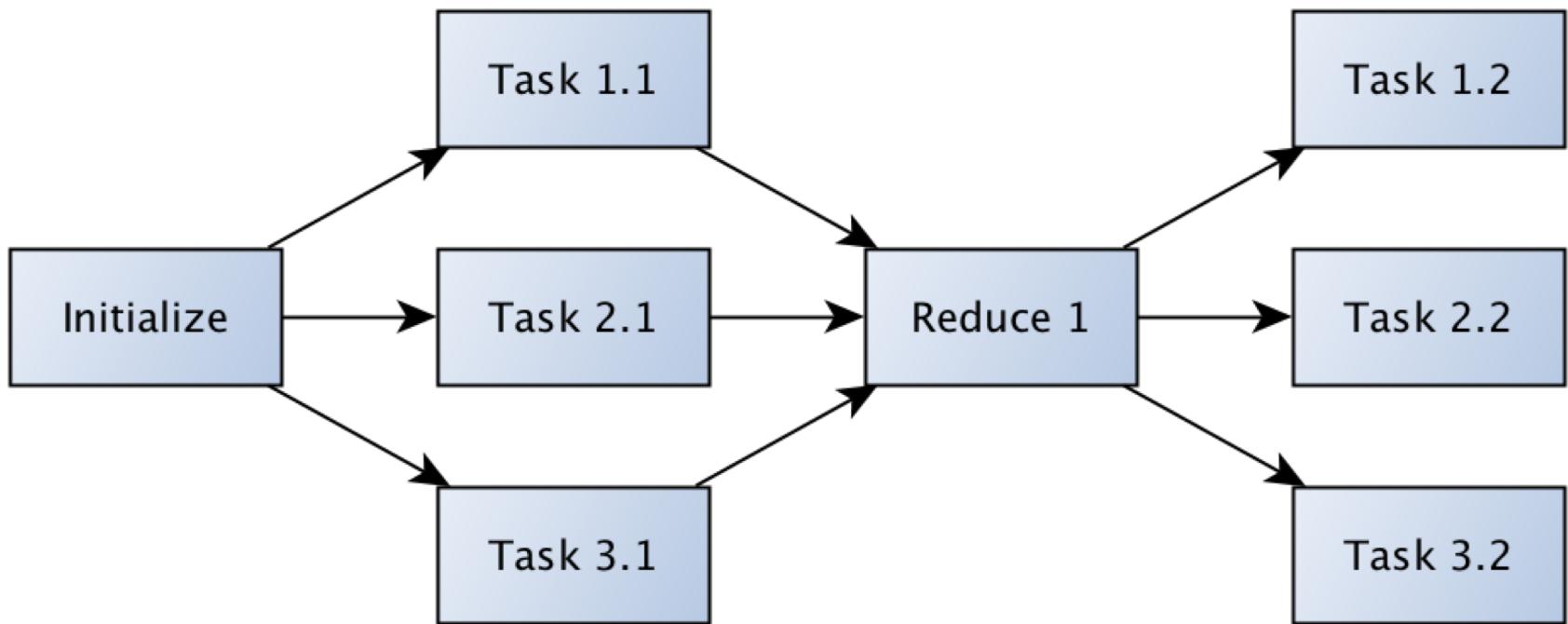
# ML на распределённых данных

---

# Map-reduce паттерн



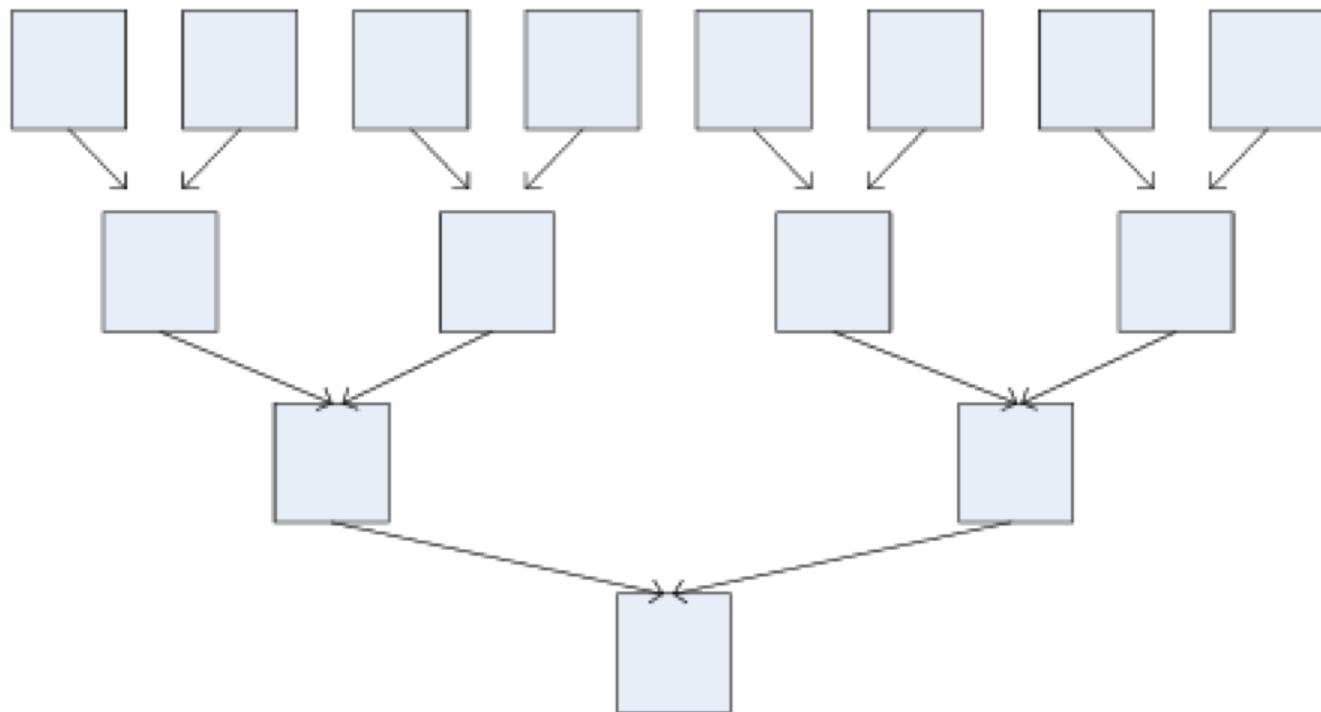
# Градиентный спуск на MapReduce



# Градиентный спуск на MapReduce: проблемы

- Драйвер – узкое место
  - Даём много памяти
  - Стаемся использовать treeReduce
- Финиш по самому медленному
  - Настроить locality.wait
  - Настроить speculation
- Время итерации >0.2s

# Tree-reduce pattern



# Пространство для оптимизации

Стохастический градиентный спуск

- Итерация дешёвая
- Итераций очень много

Полный градиентный спуск

- Дорогая итерация
- Итераций мало

Пакетный градиентный спуск

- Больше пакет – дороже итерация
- Меньше пакет – больше итераций

# Размер имеет значение



# Размер имеет значение

Большим  
данным нужен  
распределенный  
стек

Итеративный  
MapReduce  
ограничивает  
размер модели

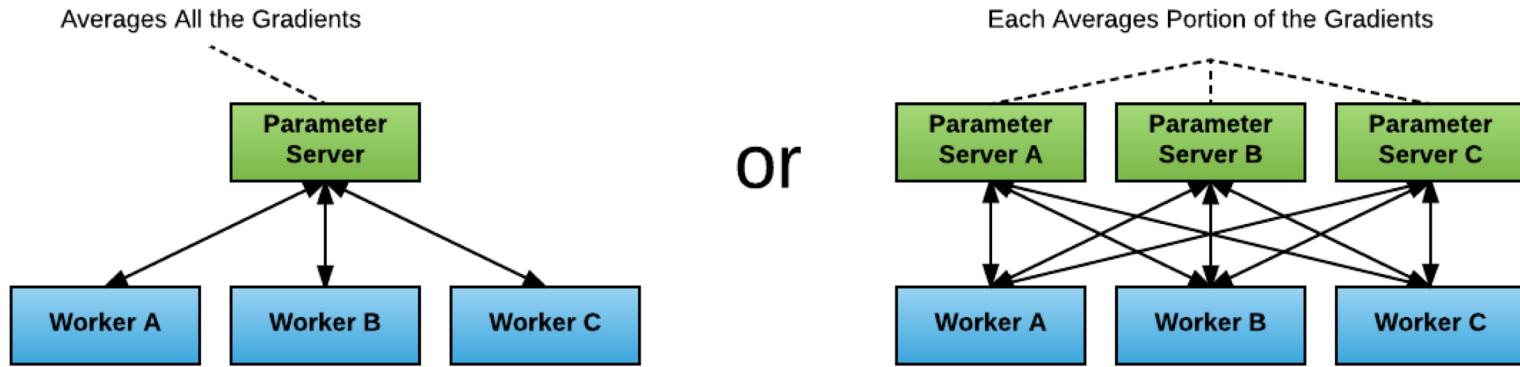
Небольшой  
модели не нужны  
большие данные



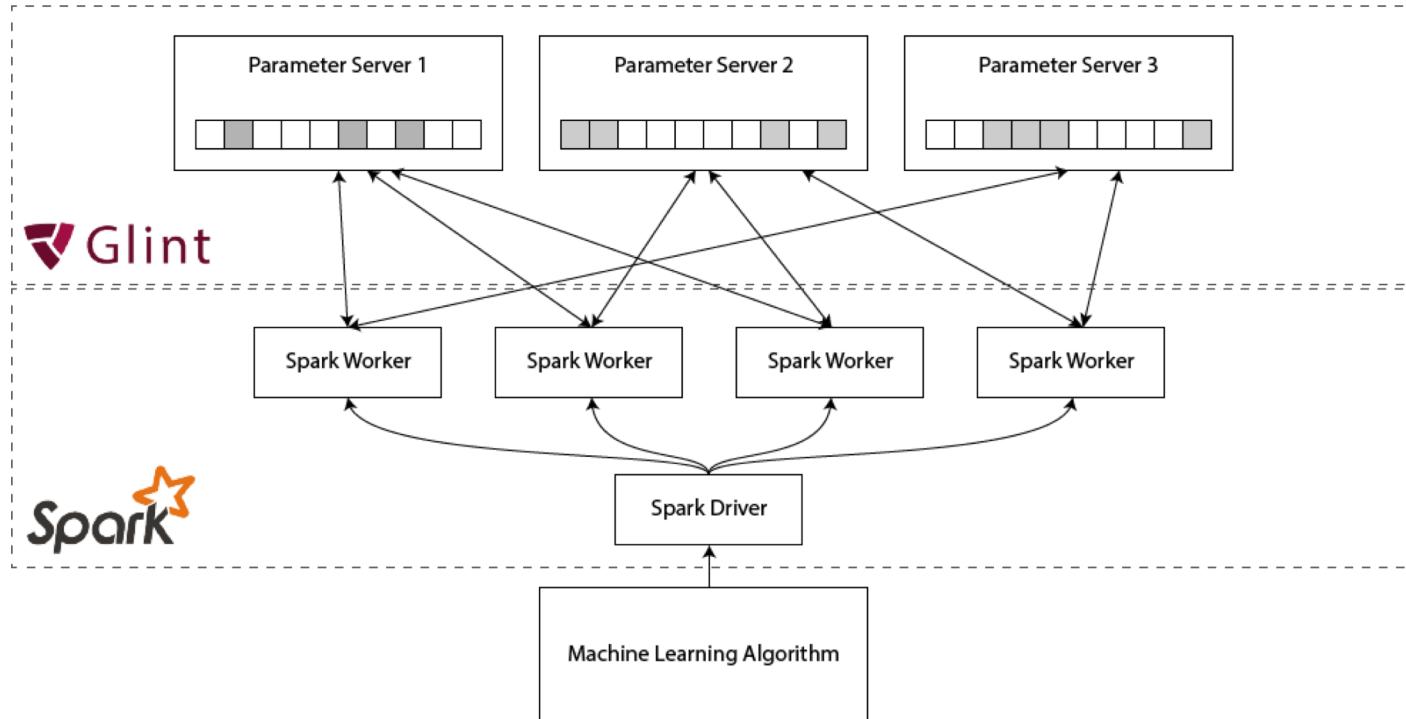
# Паттерн-который-нельзя-называть

1. Берем большие данные
2. Применяем распределенный ETL
3. Получаем маленькие данные
4. Применяем централизованный МЛ
5. Получаем качественную модель

# Parameter server – enlarge your model



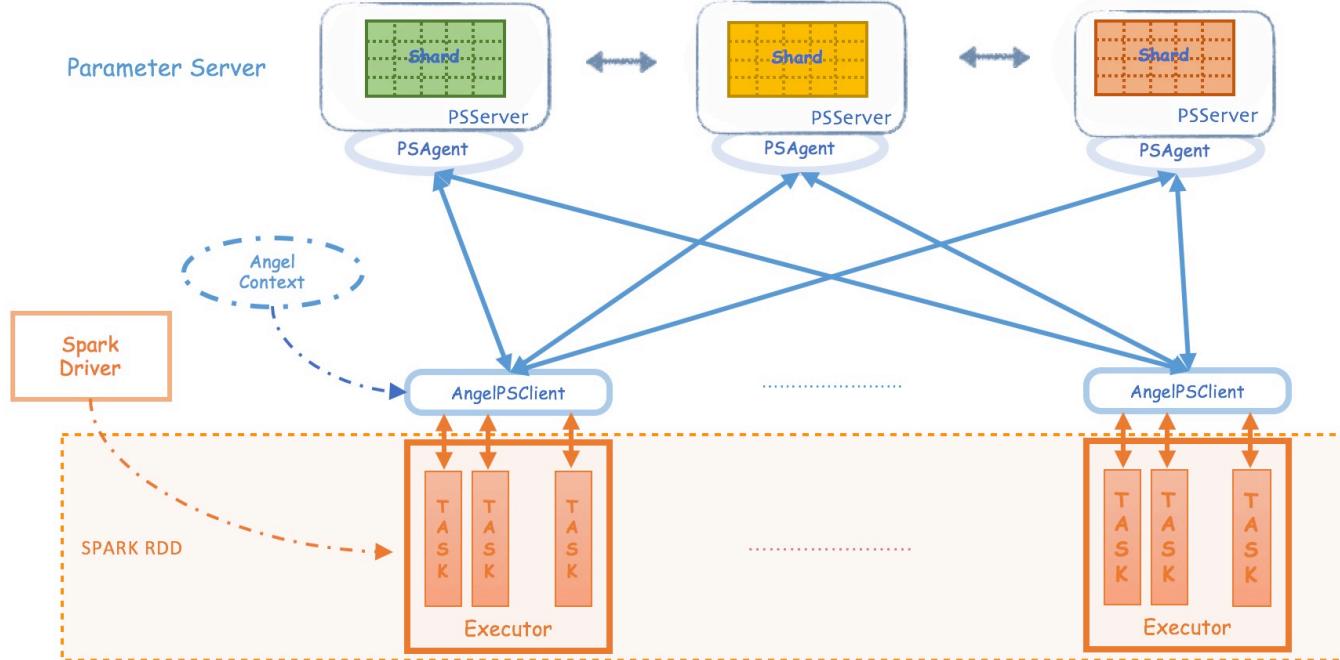
# Glint



# Glint

- Создан в датабрикс
- Работает поверх Akka
- Простые операции:
  - Создать вектор/матрицу
  - Получить кусок
  - Запушить аддитивную дельту
- 3 года не было коммитов 😞

# Angel-PS



# Angel-PS

- Разрабатывается в Tencent
  - Много документации на китайском 😞
- Активно развивается как часть платформы Angel-ML
- Те же простые операции
- Выглядит как industry-ready 😊

## Еще варианты

- H2O distributed key-value
- Apache Ignite
- PS2
- ...

# Barrier execution mode

## Обычный планировщик

- Задачи стартуют независимо
- Задачи завершаются независимо
- Задачи рестартуют независимо

## Gang scheduler (Spark 2.4)

- Задачи стартуют вместе
- Задачи завершаются вместе
- Рестартуют тоже всей бандой
- Идеально для Parameter Server

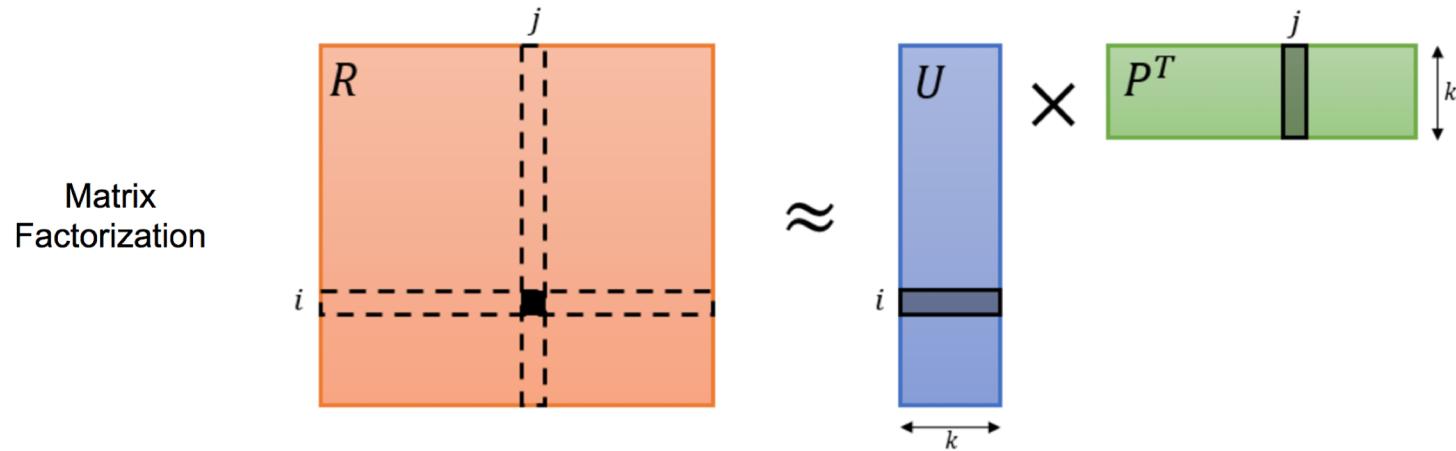
# Parameter server

- Модели до нескольких гигабайт
- Добавляет головняка с гонками
- Хорошо работает с «распределёнными» моделями
  - Каждый апдейт затрагивает только часть параметров

# ML на распределённых моделях

---

# Факторизация матриц



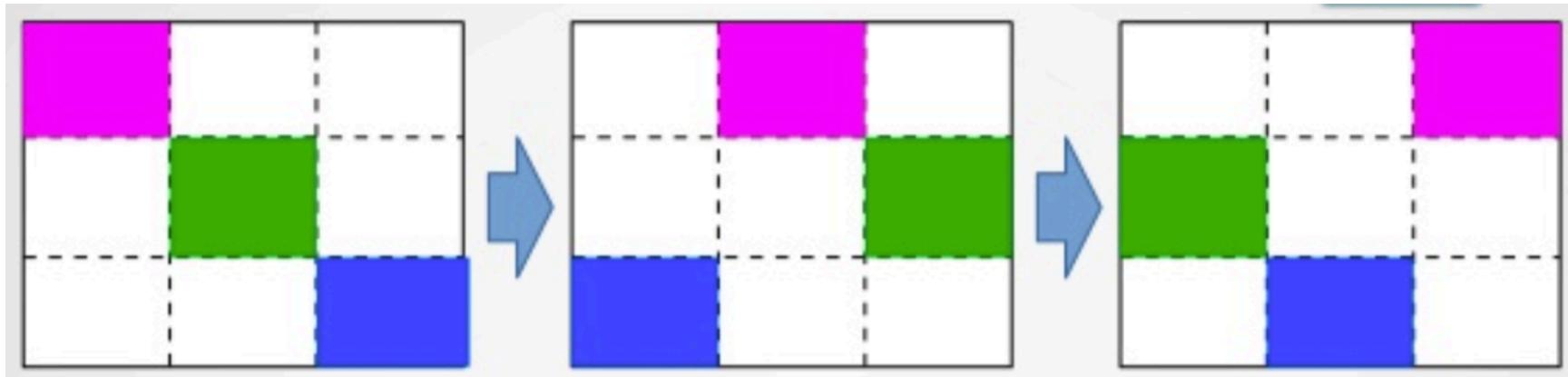
Cost Function

$$J = \|R - U \times P^T\|_2 + \lambda (\|U\|_2 + \|P\|_2)$$

# Alternating Least Squares



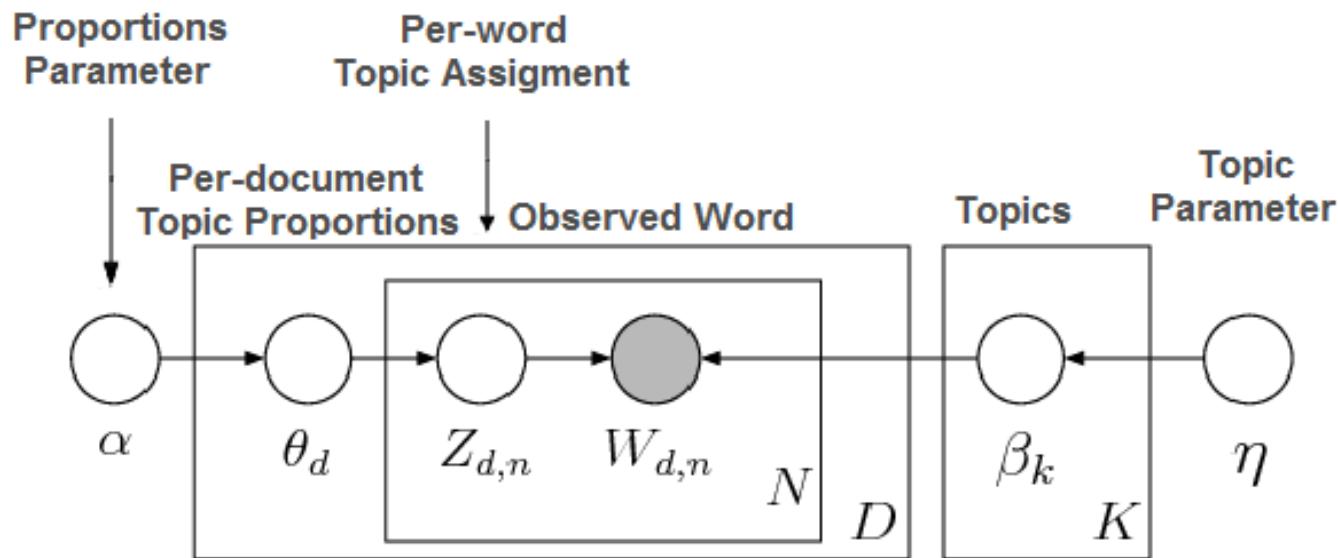
# Распределенный блочный SVD



# Распределенный блочный SVD: пример

1. Данные нарезаем на столбцы и колонки
2. Кешируем на экзекуторах блоки на пересечениях колонки/столбца с избыточностью
3. Факторами управляем через параметр сервер
4. Задачи раздаем оркестратором

# Латентные алокации Дирихле: LDA



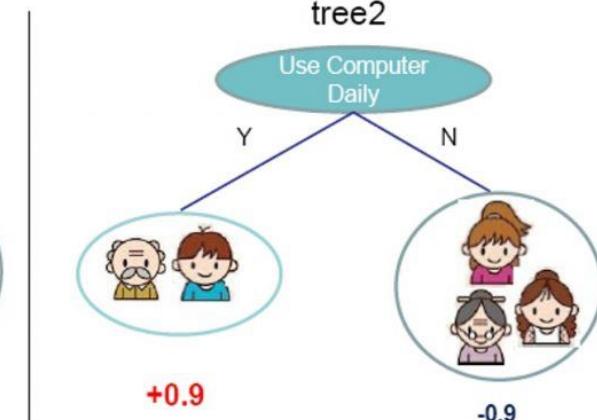
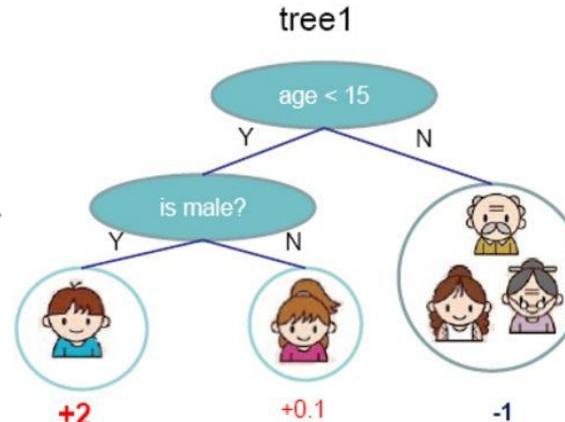
# LDA: join модели и данных

- Имеем: большой корпус и небольшой словарь
- Ищем: тематическую модель для слов и документов
- Комбинируем:
  - Топики документа обновляем независимо от остальных
  - Распределение слов по топикам храним на параметр сервере

# Деревья принятия решений

Input: age, gender, occupation, ...

Like the computer game X



prediction score in each leaf

$$f(\text{boy}) = 2 + 0.9 = 2.9$$

$$f(\text{old man}) = -1 + 0.9 = -0.1$$

<https://xgboost.readthedocs.io/en/latest/tutorials/model.html>

# Деревья – кладезь параллелизма

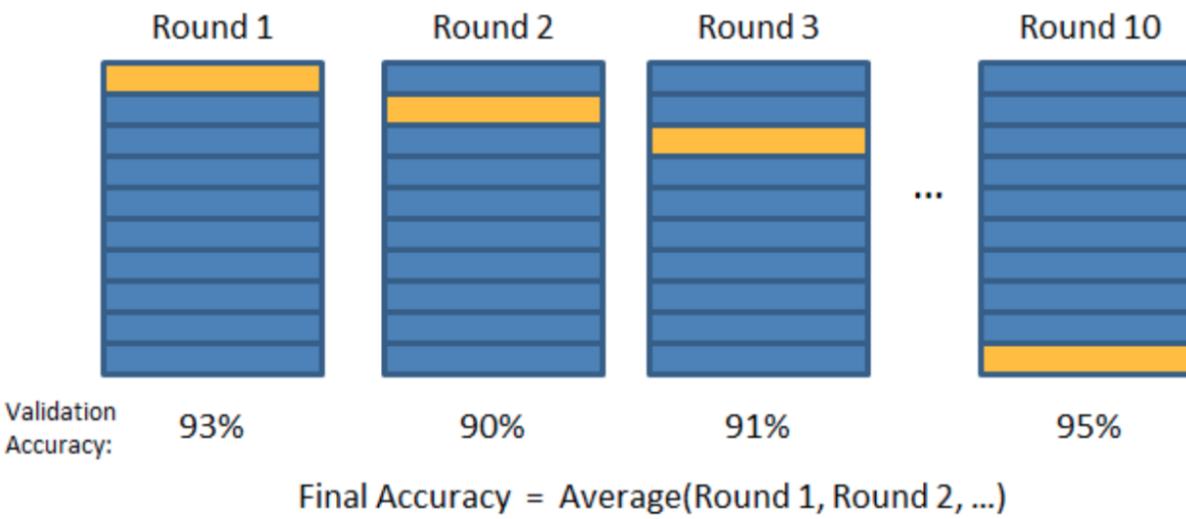
- Поиск сплита в узле
  - Каждую колонку можно рассматривать независимо
- Построение поддеревьев
  - Поддеревья строим независимо друг от друга
- Выращивание леса
  - При бэгинге деревья независимы
  - При бустинге деревья растут только последовательно

# ML на распределённых задачах

---

# Кросс-валидация

 Validation Set  
 Training Set



# Кросс-валидация

- Нет коммуникации между фолдами при обучении
- Нет разделяемого изменяющего состояния
- Большое пересечение по данным для обучения

# Кросс-валидация

- Нет коммуникации между фолдами при обучении
- Нет разделяемого изменяемого состояния
- Большое пересечение по данным для обучения
- **Идеальный кандидат на запуск в распределённой среде**

# 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()
```

- Последовательно по фолду
- Каждый фолд кешируется отдельно и безусловно

# 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()
```

- Последовательно по фолду
- Каждый фолд кешируется отдельно и безусловно



# Источники параллелизма уровня задачи

- Кросс-валидация
- Отбор признаков
- Мульти-классовая классификация one-vs-rest
- Ансамбли по принципу бэгинга
- ...

Программистам  
мало платят...



Зачем?

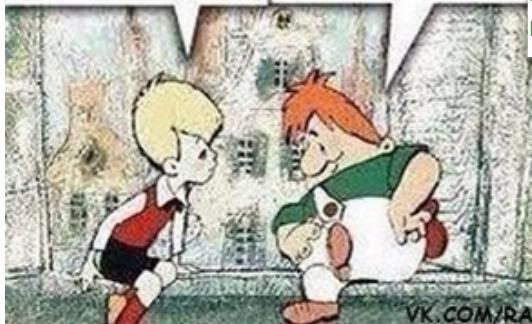
Делать AI  
за 300 К/с!

А ты становись дата  
саентистом, как я!



Ты че пес, я  
математик!

Ты же просто переби-  
раешь параметры по-  
ка валидация не даст  
нужный результат!



# Источники параллелизма уровня задачи

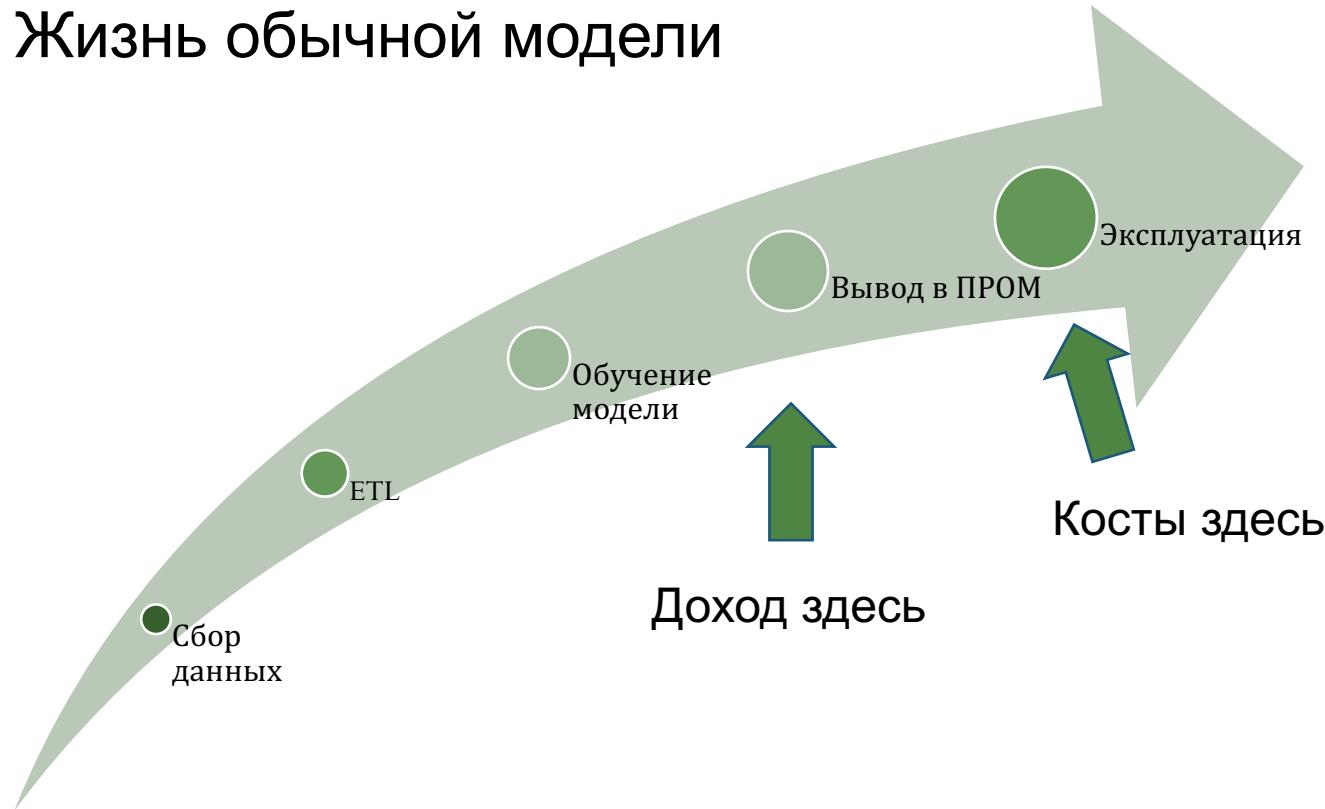
- Кросс-валидация
- Генерация и отбор признаков
- Мульти-классовая классификация one-vs-rest
- Ансамбли по принципу бэгинга
- **Подбор гиперпараметров**
- ...

# Key take aways

- Избегать распределенного МЛ
  - Наращивать железо и уменьшать данные
  - Распределенный ETL и централизованный ML
- Небольшие модели тренировать MapReduce
  - Улучшать качество отбором признаков и тюнингом параметров
- Модели крупнее резать на части
  - Параметр сервер + all-reduce в помощь
- Деревья распределяются круче сеток

# Вместо заключения: больше лучше?

## Жизнь обычной модели



# Кто разрабатывает распределенный ML?

---

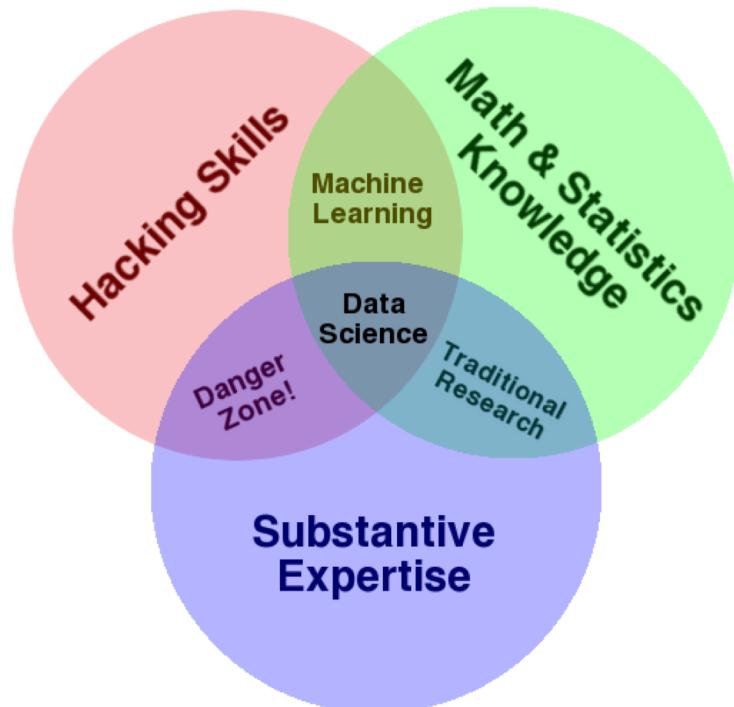
# Что нужно для переноса МЛ в распределенную среду?

- Понимать математическую основу алгоритма
  - Уметь при необходимости её модифицировать
- Знать и иметь возможности технологий распределенной обработки данных
  - Уметь при необходимости их дорабатывать
- Понимать особенности работы алгоритмов в распределенной среде
  - Уметь искать компромисс цена/качество

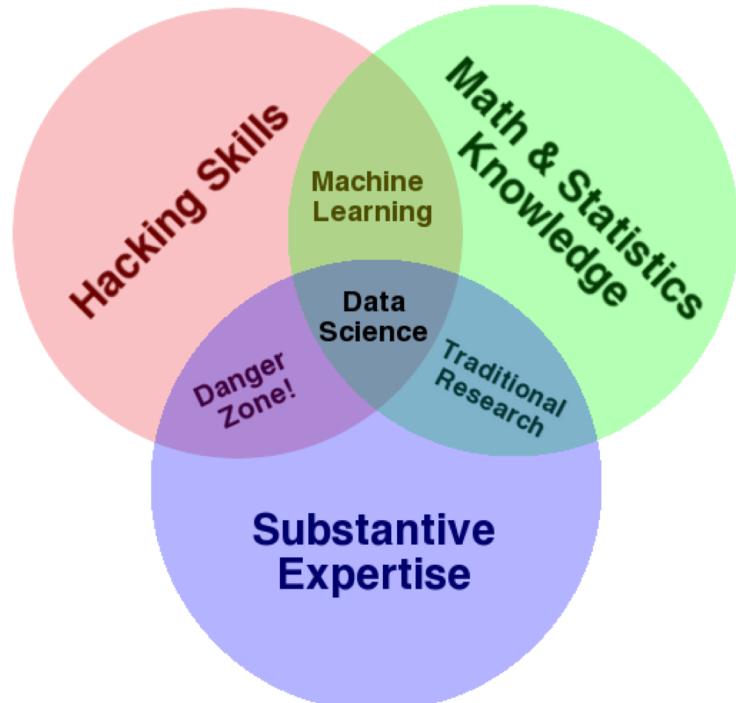
# Flashback: Основные понятия

- Data Mining
  - Поиск **неочевидных** зависимостей в данных для использования в **практических целях**.
- Machine Learning
  - Семейство **инструментов** поиска зависимостей в данных через **подбор параметров** некоторой **математической модели**.

# Data Science: Drew Conway, 2010

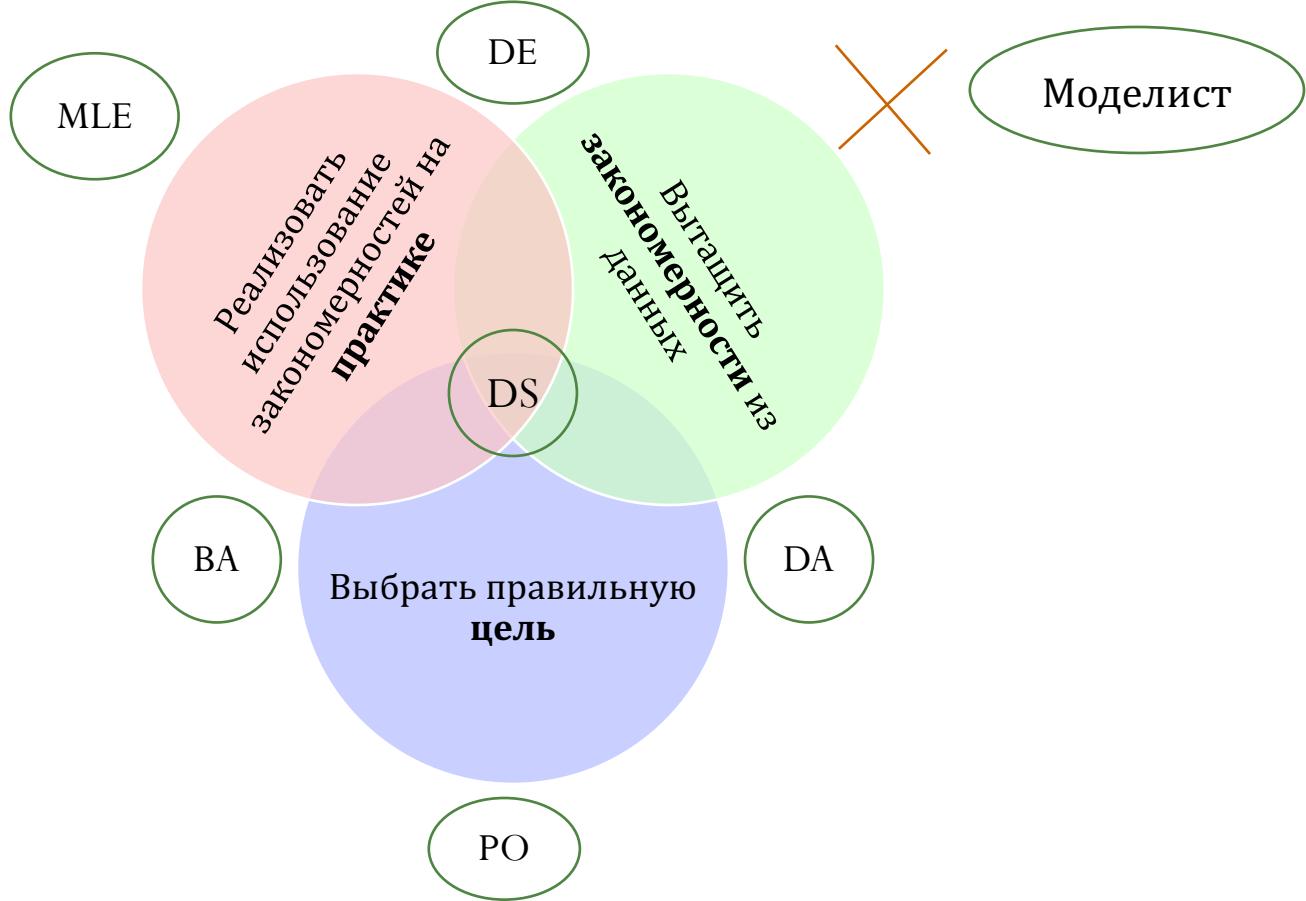


# Data Science: Drew Conway, 2010

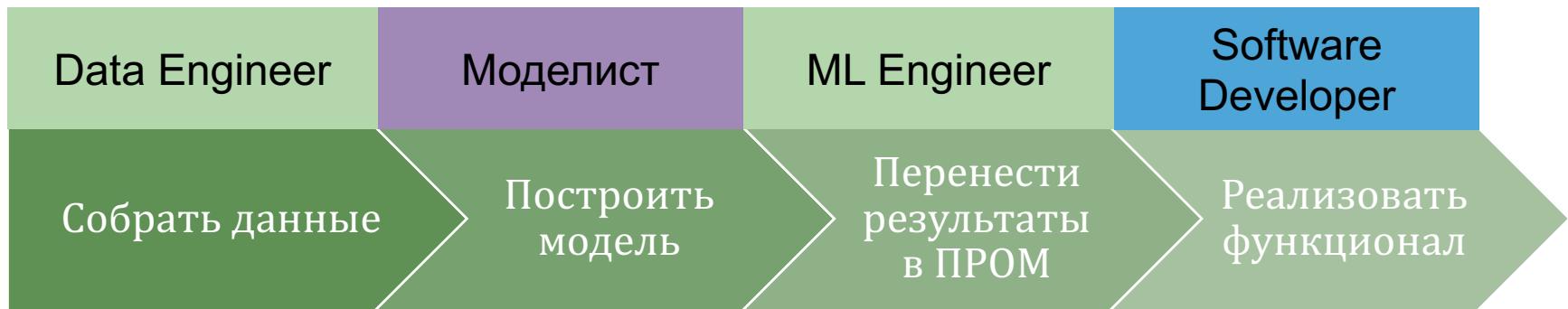


Data Scientist 2010-го года справиться с  
разработкой распределенного ML!

# Data Science: 2020



# Data Science: 2020



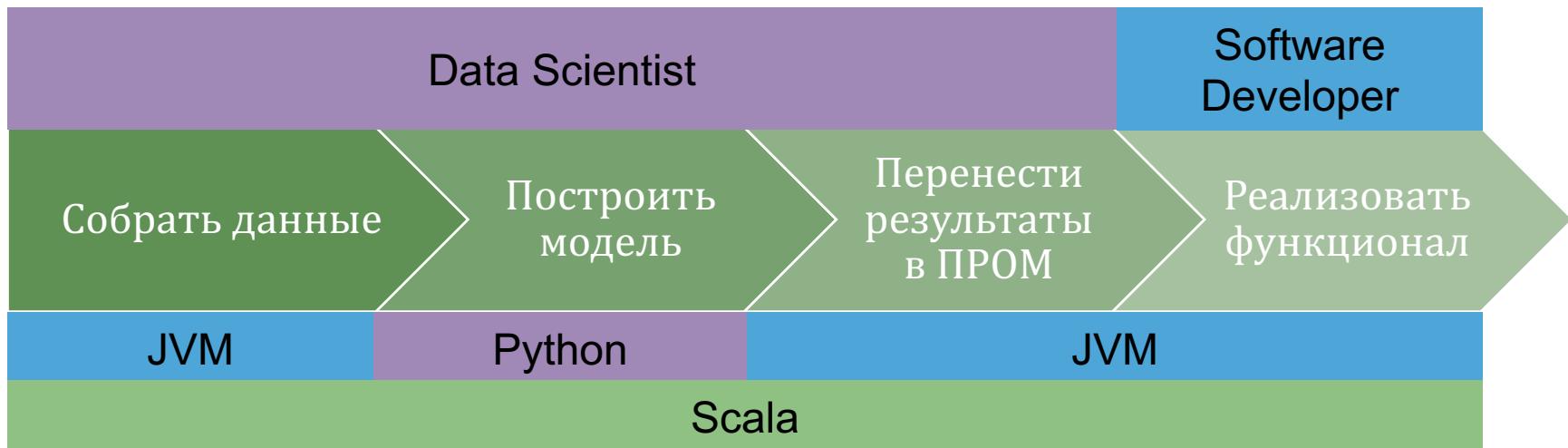
# Make Data Science Great Again!



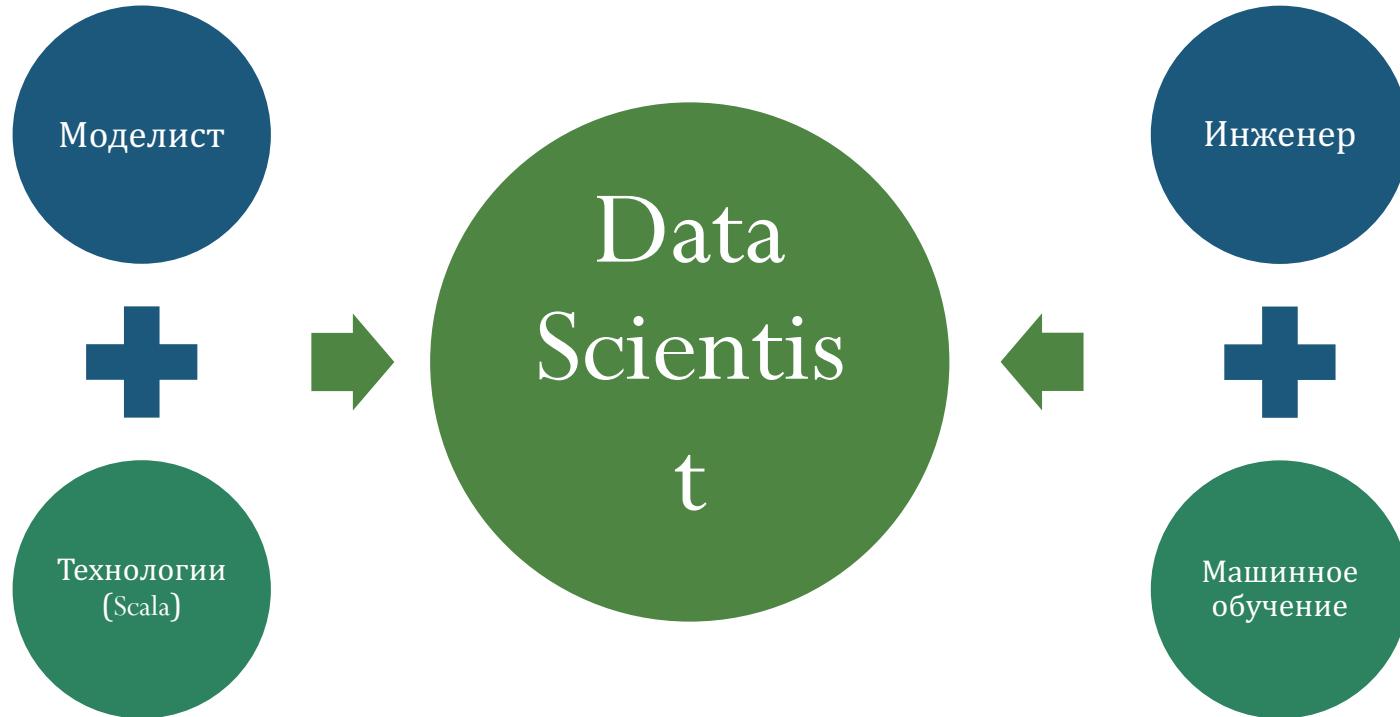
# Make Data Science Great Again!



# Make Data Science Great Again!



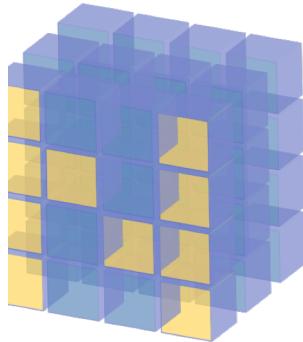
# Data Science: Два пути к цели



# Scala for Data Science

---

# Data Mining инструменты: Python



NumPy

pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$

matplotlib



# Data Mining инструменты: Scala



SQL  
MLlib

# Breeze: ключевые типы

- `Tensor[K,V]`
  - `Vector[V] : Tensor[Int,V]`
    - `DenseVector[V]`
    - `SparseVector[V]`
    - `HashVector[V]`
  - `Matrix[V] : Tensor[(Int,Int),V]`
    - `DenseMatrix[V]`
    - `CSCMatrix[V]`

# Breeze: создание объектов

```
import breeze.linalg._  
import breeze.numerics._
```

FINISHED ▶ ✎ 📄 ⚙

```
import breeze.linalg._  
import breeze.numerics._
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
from numpy import *
```

```
println(DenseMatrix.zeros[Double](3,5))  
println(DenseVector.fill(3){5.0})  
println(DenseMatrix((1.0,2.0), (3.0,4.0)))
```

FINISHED ▶ ✎ 📄 ⚙

```
0.0 0.0 0.0 0.0 0.0  
0.0 0.0 0.0 0.0 0.0  
0.0 0.0 0.0 0.0 0.0  
DenseVector(5.0, 5.0, 5.0)  
1.0 2.0  
3.0 4.0
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
print(zeros((3,5)))  
print(ones(3) * 5)  
print(array([[ 1,2], [3,4]]))
```

```
[[0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0.]  
 [0. 0. 0. 0. 0.]]  
[5. 5. 5.]  
[[1 2]  
 [3 4]]
```

# Breeze: создание объектов

```
println(linspace(2,7,4))
println(DenseMatrix.tabulate(3, 2){case (i, j) => i+j})
println(DenseMatrix.rand(2, 3))
```

```
DenseVector(2.0, 3.666666666666667, 5.333333333333334, 7.0)
```

```
0 1
```

```
1 2
```

```
2 3
```

```
0.0011384085492853746 0.8906803221697828 0.28292512196249686
```

```
0.6933233567133361 0.9519212175449716 0.2189689656138083
```

# Breeze: Слайсинг и индексация

FINISHED ▶ ✎ 📄 ⚙

```
val m = DenseMatrix.tabulate(3, 3){case (i, j) => i-j}
val cat = m(::, 1 to 2)
val e = m(-1, 1)

m: breeze.linalg.DenseMatrix[Int] =
0 -1 -2
1  0  -1
2  1   0
cat: breeze.linalg.DenseMatrix[Int] =
-1 -2
0  -1
1  0
e: Int = 1
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:34:26 PM.

%python

```
m = array([[0,-1,-2],[1,0,-1],[2,1,0]])
print(m)
print(m[:,1:3])
print(m[-1,1])

[[ 0 -1 -2]
 [ 1  0 -1]
 [ 2  1  0]]
[[-1 -2]
 [ 0 -1]
 [ 1  0]]
1
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:34:26 PM.

# Breeze: Слайсинг и индексация

FINISHED ▶ ✎ 📄 ⚙

```
val m = DenseMatrix.tabulate(3, 3){case (i, j) => i-j}
val cat = m(::, 1 to 2)
val e = m(-1, 1)
```

```
m: breeze.linalg.DenseMatrix[Int] =
0 -1 -2
1  0  -1
2  1   0
cat: breeze.linalg.DenseMatrix[Int] =
[-1 -2]
[0 -1]
[1  0]
e: Int = 1
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:34:26 PM.

%python

```
m = array([[0,-1,-2],[1,0,-1],[2,1,0]])
print(m)
print(m[:,1:3])
print(m[-1,1])
```

```
[[ 0 -1 -2]
 [ 1  0 -1]
 [ 2  1  0]]
[[-1 -2]
 [ 0 -1]
 [ 1  0]]
1
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:34:26 PM.

# Breeze: Слайсинг и индексация

FINISHED ▶ ✎ 📄 ⚙

```
val m = DenseMatrix.tabulate(3, 3){case (i, j) => i-j}
val cat = m(::, 1 to 2)
val e = m(-1, 1)
```

```
m: breeze.linalg.DenseMatrix[Int] =
0 -1 -2
1  0  -1
2  1   0
cat: breeze.linalg.DenseMatrix[Int] =
-1 -2
0  -1
1   0
e: Int = 1
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:34:26 PM.

%python

```
m = array([[0,-1,-2],[1,0,-1],[2,1,0]])
print(m)
print(m[:,1:3])
print(m[-1,1])
```

```
[[ 0 -1 -2]
 [ 1  0 -1]
 [ 2  1  0]]
[[-1 -2]
 [ 0 -1]
 [ 1  0]]
1
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:34:26 PM.

# Breeze: Умножение матриц

```
{  
    val v = DenseVector(1.0, 2.0, 3.0)  
    val A = DenseMatrix((1.0,2.0,3.0), (1.0,0.0,2.0))  
    println(A * v)  
    println(v dot v)  
    println(v.t * v)  
    println(A * A.t)  
}  
  
DenseVector(14.0, 7.0)
```

14.0  
14.0  
14.0 7.0  
7.0 5.0

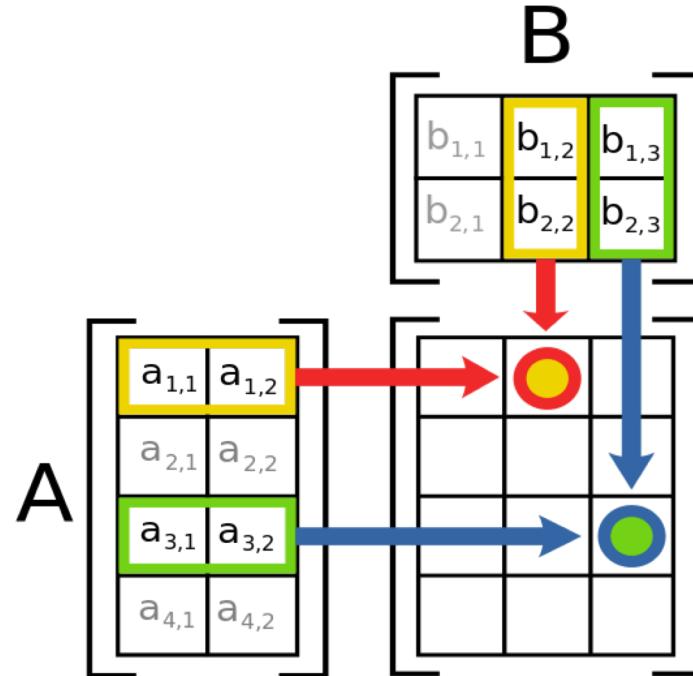
Took 1 sec. Last updated by dmitry.bugaychenko at March 31 2019, 10:39:10 PM.

FINISHED ▶ ✎ 📄 ⚙

```
%python  
v = array([1.0, 2.0, 3.0])  
A = array([[1.0,2.0,3.0],[1.0,0.0,2.0]])  
print(A.dot(v))  
print(v.dot(v))  
print(v.T.dot(v))  
print(A.dot(A.T))  
  
[14. 7.]  
14.0  
14.0  
[[14. 7.]  
 [ 7. 5.]]
```

Took 0 sec. Last updated by dmitry.bugaychenko at March 31 2019,

# Умножение матриц



# Breeze: Поэлементные операции

```
{  
    val v = DenseVector(1.0, 2.0, 3.0)  
    println(v *:* v)  
    v += v  
    println(v)  
    println(v :*= 3.0)  
    println(v <:< (v +:+ v))  
    println(argmax(v))  
}  
  
DenseVector(1.0, 4.0, 9.0)  
DenseVector(2.0, 4.0, 6.0)  
DenseVector(6.0, 12.0, 18.0)  
BitVector(0, 1, 2)
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
v = array([1.,2.,3.])  
print(v * v)  
v += v  
print(v)  
v *= 3  
print(v)  
print(v < (v + v))  
print(argmax(v))
```

```
[1. 4. 9.]  
[2. 4. 6.]  
[ 6. 12. 18.]  
[ True True True]  
2
```

# Breeze: Поэлементные операции

```
{  
    val v = DenseVector(1.0, 2.0, 3.0)  
    println(v *:  
    v := v  
    println(v)  
    println(v := 3.0)  
    println(v <:< (v +:+ v))  
    println(argmax(v))  
}  
  
DenseVector(1.0, 4.0, 9.0)  
DenseVector(2.0, 4.0, 6.0)  
DenseVector(6.0, 12.0, 18.0)  
BitVector(0, 1, 2)  
2
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
v = array([1.,2.,3.])  
print(v * v)  
v += v  
print(v)  
v *= 3  
print(v)  
print(v < (v + v))  
print(argmax(v))
```

```
[1. 4. 9.]  
[2. 4. 6.]  
[ 6. 12. 18.]  
[ True  True  True]  
2
```

# Breeze: Поэлементные операции

```
{  
    val v = DenseVector(1.0, 2.0, 3.0)  
    println(v *:  
        v  
        v := v  
        println(v)  
        println(v *:= 3.0)  
        println(v <:< (v +:+ v))  
        println(argmax(v))  
    }  
  
DenseVector(1.0, 4.0, 9.0)  
DenseVector(2.0, 4.0, 6.0)  
DenseVector(6.0, 12.0, 18.0)  
BitVector(0, 1, 2)  
2
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
v = array([1.,2.,3.])  
print(v * v)  
v += v  
print(v)  
v *= 3  
print(v)  
print(v < (v + v))  
print(argmax(v))
```

```
[1. 4. 9.]  
[2. 4. 6.]  
[ 6. 12. 18.]  
[ True  True  True]  
2
```

# Breeze: Поэлементные операции

```
{  
    val v = DenseVector(1.0, 2.0, 3.0)  
    println(v *:* v)  
    v := v  
    println(v)  
    println(v :*= 3.0)  
    println(v <:< (v +:+ v))  
    println(argmax(v))  
}  
  
DenseVector(1.0, 4.0, 9.0)  
DenseVector(2.0, 4.0, 6.0)  
DenseVector(6.0, 12.0, 18.0)  
BitVector(0, 1, 2)
```

2

FINISHED ▶ ✎ 📄 ⚙

```
%python  
v = array([1.,2.,3.])  
print(v * v)  
v += v  
print(v)  
v *= 3  
print(v)  
print(v < (v + v))  
print(argmax(v))
```

```
[1. 4. 9.]  
[2. 4. 6.]  
[ 6. 12. 18.]  
[ True True True]  
2
```

# Breeze: Поэлементные операции

```
{  
    val v = DenseVector(1.0, 2.0, 3.0)  
    println(v *:  
    v  
    v := v  
    println(v)  
    println(v :*= 3.0)  
    println(v <:< (v +:+ v))  
    println(argmax(v))  
}  
  
DenseVector(1.0, 4.0, 9.0)  
DenseVector(2.0, 4.0, 6.0)  
DenseVector(6.0, 12.0, 18.0)  
BitVector(0, 1, 2)  
2
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
v = array([1.,2.,3.])  
print(v * v)  
v += v  
print(v)  
v *= 3  
print(v)  
print(v < (v + v))  
print(argmax(v))
```

```
[1. 4. 9.]  
[2. 4. 6.]  
[ 6. 12. 18.]  
[ True  True  True]  
2
```

# Breeze: Агрегаты

```
{  
  
    val A = DenseMatrix((1.0,2.0,3.0), (1.0,0.0,2.0))  
    println(sum(A))  
    println(mean(A(::, *)))  
    println(trace(A(::, 0 to 1)))  
    println(accumulate(A.toDenseVector))  
}  
  
9.0  
Transpose(DenseVector(1.0, 1.0, 2.5))  
1.0  
DenseVector(1.0, 2.0, 4.0, 4.0, 7.0, 9.0)
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
  
A = array([[1.0,2.0,3.0], [1.0,0.0,2.0]])  
print(sum(A))  
print(mean(A, 0))  
print(trace(A[:,0:2]))  
print(A.T.cumsum())  
  
9.0  
[1. 1. 2.5]  
1.0  
[1. 2. 4. 4. 7. 9.]
```

Took 0 sec. Last updated by dmitry.bugaychenko at April 01 2019, 12:3

# Breeze: Агрегаты

```
{  
  
    val A = DenseMatrix((1.0,2.0,3.0), (1.0,0.0,2.0))  
    println(sum(A))  
    println(mean(A(::, *)))  
    println(trace(A(::, 0 to 1)))  
    println(accumulate(A.toDenseVector))  
}  
  
9.0  
Transpose(DenseVector(1.0, 1.0, 2.5))  
1.0  
DenseVector(1.0, 2.0, 4.0, 4.0, 7.0, 9.0)
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
  
A = array([[1.0,2.0,3.0], [1.0,0.0,2.0]])  
print(sum(A))  
print(mean(A, 0))  
print(trace(A[:,0:2]))  
print(A.T.cumsum())  
  
9.0  
[1. 1. 2.5]  
1.0  
[1. 2. 4. 4. 7. 9.]
```

Took 0 sec. Last updated by dmitry.bugaychenko at April 01 2019, 12:3

# Breeze: Агрегаты

{

FINISHED ▶ ✎ 📄 ⚙

```
val A = DenseMatrix((1.0,2.0,3.0), (1.0,0.0,2.0))
println(sum(A))
println(mean(A:::, *)))
println(trace(A:::, 0 to 1)))
println(accumulate(A.toDenseVector))
}
```

9.0

Transpose(DenseVector(1.0, 1.0, 2.5))

1.0

DenseVector(1.0, 2.0, 4.0, 4.0, 7.0, 9.0)

%python

FINI

```
A = array([[1.0,2.0,3.0], [1.0,0.0,2.0]])
print(sum(A))
print(mean(A, 0))
print(trace(A[:,0:2]))
print(A.T.cumsum())
```

9.0

[1. 1. 2.5]

1.0

[1. 2. 4. 4. 7. 9.]

Took 0 sec. Last updated by dmitry.bugaychenko at April 01 2019, 12:3

# Breeze: Агрегаты

```
{  
  
    val A = DenseMatrix((1.0,2.0,3.0), (1.0,0.0,2.0))  
    println(sum(A))  
    println(mean(A(::, *)))  
    println(trace(A(::, 0 to 1)))  
    println(accumulate(A.toDenseVector))  
}  
  
9.0  
Transpose(DenseVector(1.0, 1.0, 2.5))  
1.0  
DenseVector(1.0, 2.0, 4.0, 4.0, 7.0, 9.0)
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
  
A = array([[1.0,2.0,3.0], [1.0,0.0,2.0]])  
print(sum(A))  
print(mean(A, 0))  
print(trace(A[:,0:2]))  
print(A.T.cumsum())  
  
9.0  
[1. 1. 2.5]  
1.0  
[1. 2. 4. 4. 7. 9.]
```

Took 0 sec. Last updated by dmitry.bugaychenko at April 01 2019, 12:3

# Breeze: Агрегаты

```
{  
  
    val A = DenseMatrix((1.0,2.0,3.0), (1.0,0.0,2.0))  
    println(sum(A))  
    println(mean(A:::, *)))  
    println(trace(A:::, 0 to 1)))  
    println(accumulate(A.toDenseVector))  
}  
  
9.0  
Transpose(DenseVector(1.0, 1.0, 2.5))  
1.0  
DenseVector(1.0, 2.0, 4.0, 4.0, 7.0, 9.0)
```

FINISHED ▶ ✎ 📄 ⚙

```
%python  
  
A = array([[1.0,2.0,3.0], [1.0,0.0,2.0]])  
print(sum(A))  
print(mean(A, 0))  
print(trace(A[:,0:2]))  
print(A.T.cumsum())  
  
9.0  
[1. 1. 2.5]  
1.0  
[1. 2. 4. 4. 7. 9.]
```

Took 0 sec. Last updated by dmitry.bugaychenko at April 01 2019, 12:3

# Breeze: Линейная алгебра

```
{  
    val A = DenseMatrix((1.0,2.0), (5.0,3.0))  
    val b = A * DenseVector(3.0,1.0)  
  
    println(A \ b)  
    println(det(A))  
    println(inv(A))  
    println(eig(A))  
}  
DenseVector(3.0, 1.0)
```

```
-7.0  
-0.42857142857142866  0.28571428571428575  
0.7142857142857143   -0.14285714285714288  
Eig(DenseVector(-1.3166247903553998, 5.3166247903554), Dense  
Vector(0.0, 0.0), -0.6534848467402069  -0.42039402943737053  
0.7569395980399812   -0.9073416445933755  )
```

```
%python  
A = array([[1.0,2.0],[5.0,3.0]])  
b = A.dot(array([3.0,1.0]))
```

```
print(linalg.solve(A,b))  
print(linalg.det(A))  
print(linalg.inv(A))  
print(linalg.eig(A))
```

```
[3. 1.]  
-6.999999999999999  
[-0.42857143  0.28571429]  
[ 0.71428571 -0.14285714]  
(array([-1.31662479,  5.31662479]), array([-0.65348485, -  
0.42039403],  
 [ 0.7569396 , -0.90734164]))
```

Took 0 sec. Last updated by dmitry.bugaychenko at April 01 2019, 1:03:00 AM. (outdated)

# Breeze: оптимизация

```
{  
    val X = DenseMatrix.rand(2000, 3)  
    val y = X * DenseVector(0.5, -0.1, 0.2)  
  
    val J = new DiffFunction[DenseVector[Double]] {  
        def calculate(w: DenseVector[Double]) = {  
            val e = X * w - y  
            val loss = sum(e ^:^ 2.0) / (2 * X.rows)  
            val grad = (e.t * X) /:/ (2.0 * X.rows)  
            (loss, grad.t)  
        }  
    }  
  
    val optimizer = new LBFGS[DenseVector[Double]]()  
    println(optimizer.minimize(J, DenseVector(0.0, 0.0, 0.0)))  
}
```

DenseVector(0.5000000161223864, -0.0999999855730413, 0.1999999757838036)

# Breeze: оптимизация

```
{  
    val X = DenseMatrix.rand(2000, 3)  
    val y = X * DenseVector(0.5, -0.1, 0.2)  
  
    val J = new DiffFunction[DenseVector[Double]] {  
        def calculate(w: DenseVector[Double]) = {  
            val e = X * w - y  
            val loss = sum(e ^:^ 2.0) / (2 * X.rows)  
            val grad = (e.t * X) /:/ (2.0 * X.rows)  
            (loss, grad.t)  
        }  
    }  
  
    val optimizer = new LBFGS[DenseVector[Double]]()  
    println(optimizer.minimize(J, DenseVector(0.0, 0.0, 0.0)))  
}
```

DenseVector(0.5000000161223864, -0.0999999855730413, 0.1999999757838036)

FINIS

# Breeze: оптимизация

```
{  
    val X = DenseMatrix.rand(2000, 3)  
    val y = X * DenseVector(0.5, -0.1, 0.2)  
  
    val J = new DiffFunction[DenseVector[Double]] {  
        def calculate(w: DenseVector[Double]) = {  
            val e = X * w - y  
            val loss = sum(e ^:^ 2.0) / (2 * X.rows)  
            val grad = (e.t * X) /:/ (2.0 * X.rows)  
            (loss, grad.t)  
        }  
    }  
  
    val optimizer = new LBFGS[DenseVector[Double]]()  
    println(optimizer.minimize(J, DenseVector(0.0, 0.0, 0.0)))  
}
```

DenseVector(0.5000000161223864, -0.0999999855730413, 0.1999999757838036)

# Breeze: оптимизация

```
{  
    val X = DenseMatrix.rand(2000, 3)  
    val y = X * DenseVector(0.5, -0.1, 0.2)  
  
    val J = new DiffFunction[DenseVector[Double]] {  
        def calculate(w: DenseVector[Double]) = {  
            val e = X * w - y  
            val loss = sum(e ^:^ 2.0) / (2 * X.rows)  
            val grad = (e.t * X) /:/ (2.0 * X.rows)  
            (loss, grad.t)  
        }  
    }  
  
    val optimizer = new LBFGS[DenseVector[Double]]()  
    println(optimizer.minimize(J, DenseVector(0.0, 0.0, 0.0)))  
}
```

DenseVector(0.5000000161223864, -0.0999999855730413, 0.1999999757838036)

FINIS

# Breeze: оптимизация

```
{  
    val X = DenseMatrix.rand(2000, 3)  
    val y = X * DenseVector(0.5, -0.1, 0.2)  
  
    val J = new DiffFunction[DenseVector[Double]] {  
        def calculate(w: DenseVector[Double]) = {  
            val e = X * w - y  
            val loss = sum(e ^:^ 2.0) / (2 * X.rows)  
            val grad = (e.t * X) /:/ (2.0 * X.rows)  
            (loss, grad.t)  
        }  
    }  
  
    val optimizer = new LBFGS[DenseVector[Double]]()  
    println(optimizer.minimize(J, DenseVector(0.0, 0.0, 0.0)))  
}
```

DenseVector(0.5000000161223864, -0.0999999855730413, 0.1999999757838036)

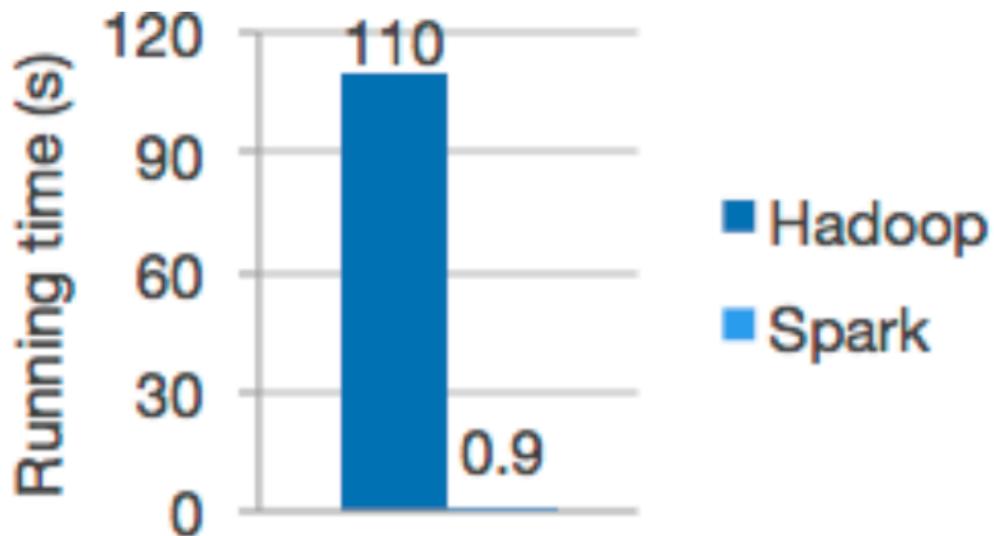
# Breeze: оптимизация

```
{  
    val X = DenseMatrix.rand(2000, 3)  
    val y = X * DenseVector(0.5, -0.1, 0.2)  
  
    val J = new DiffFunction[DenseVector[Double]] {  
        def calculate(w: DenseVector[Double]) = {  
            val e = X * w - y  
            val loss = sum(e ^:^ 2.0) / (2 * X.rows)  
            val grad = (e.t * X) /:/ (2.0 * X.rows)  
            (loss, grad.t)  
        }  
    }  
  
    val optimizer = new LBFGS[DenseVector[Double]]()  
    println(optimizer.minimize(J, DenseVector(0.0, 0.0, 0.0)))  
}
```

DenseVector(0.5000000161223864, -0.0999999855730413, 0.1999999757838036)

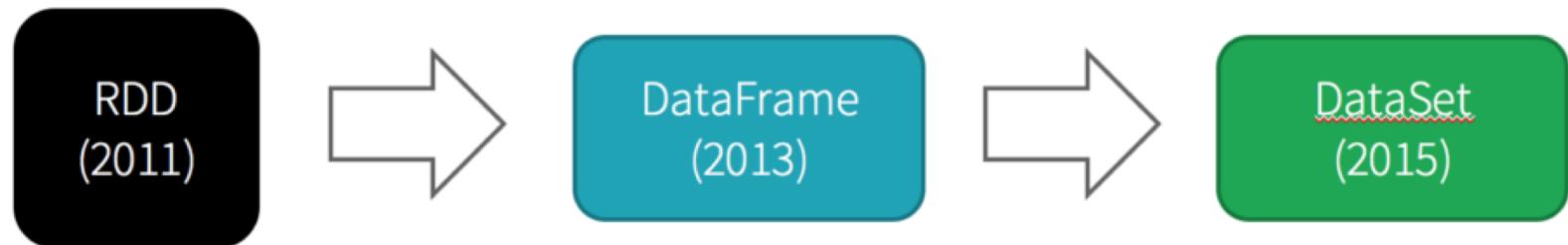
# Spark ML

---



Logistic regression in Hadoop and Spark

# History of Spark APIs



Distribute collection  
of JVM objects

Functional Operators (map,  
filter, etc.)

Distribute collection  
of Row objects

Expression-based operations  
and UDFs

Logical plans and optimizer

Fast/efficient internal  
representations

Internally rows, externally  
JVM objects

Almost the “Best of both  
worlds”: type safe + fast

But slower than DF  
Not as good for interactive  
analysis, especially Python

# Spark MLlib



## RDD API

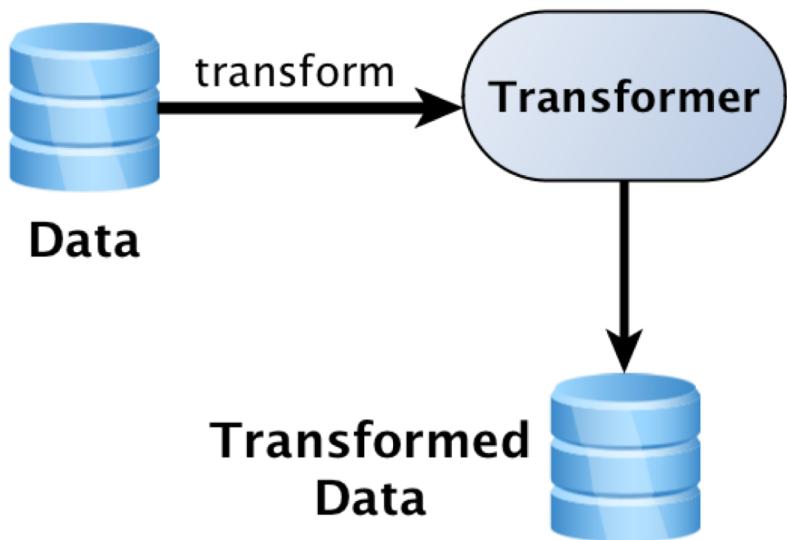
- Атомарные алгоритмы
- Не будет развиваться

## Dataset API

- Комплексные конвейеры
- Часто RDD API под капотом

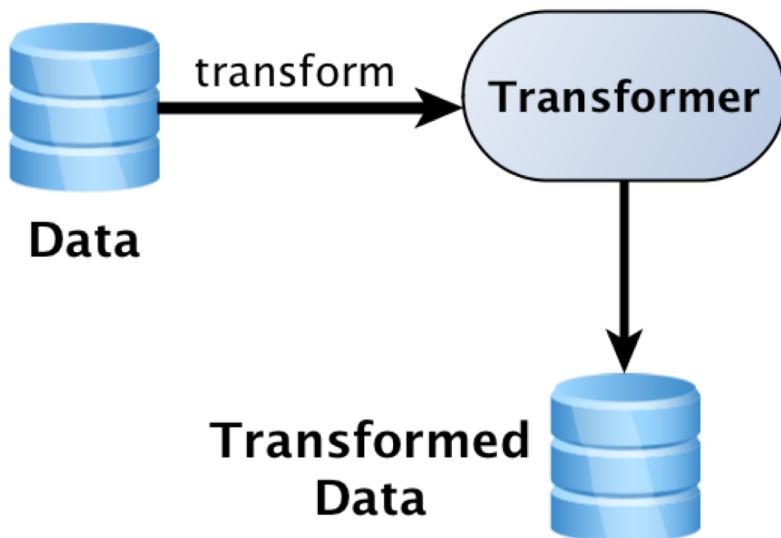
# Spark ML Pipelines

## Transformer

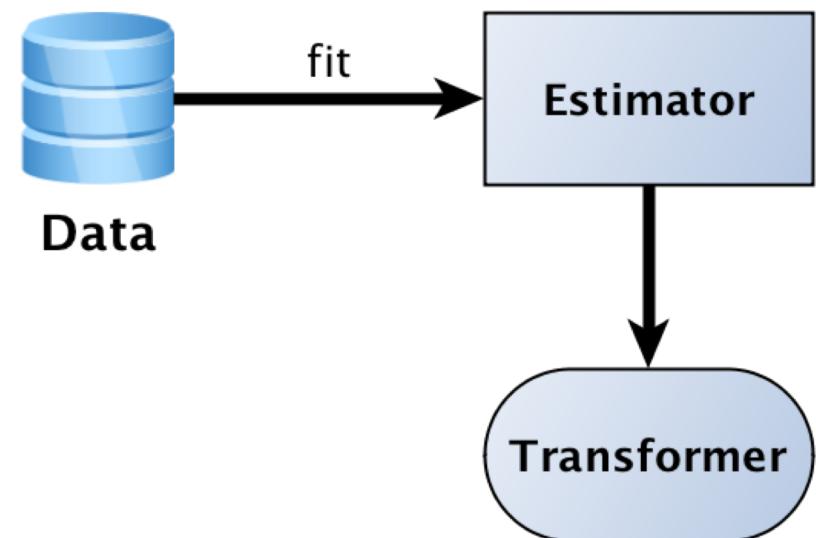


# Spark ML Pipelines

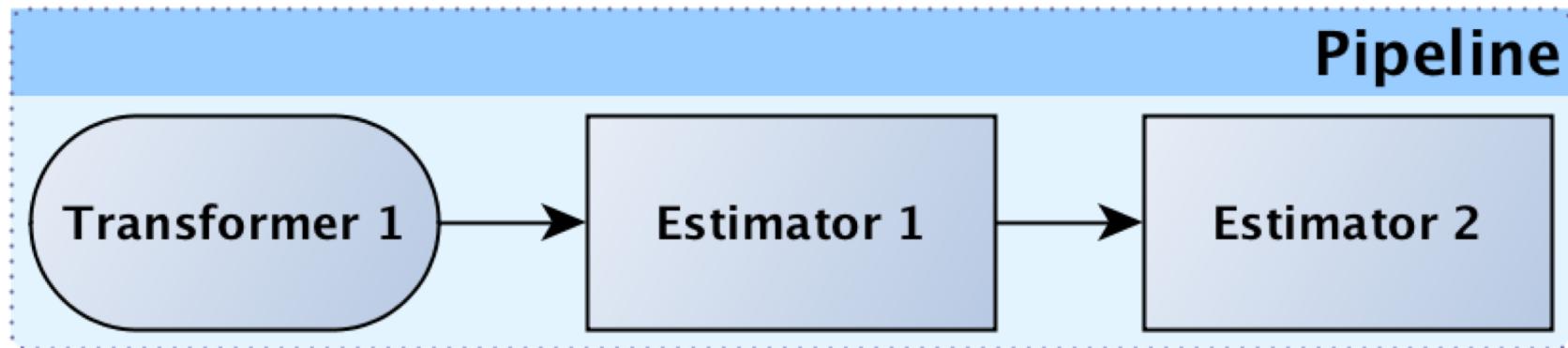
Transformer



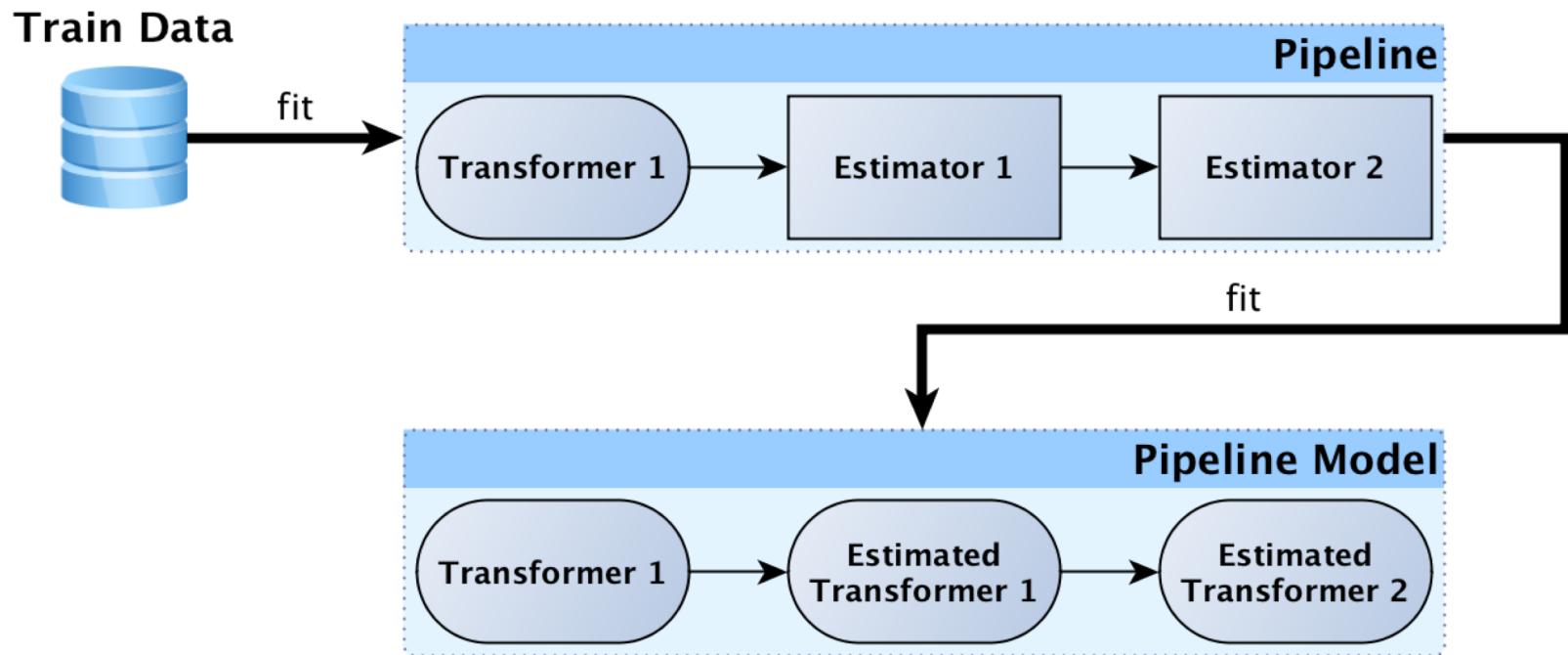
Estimator



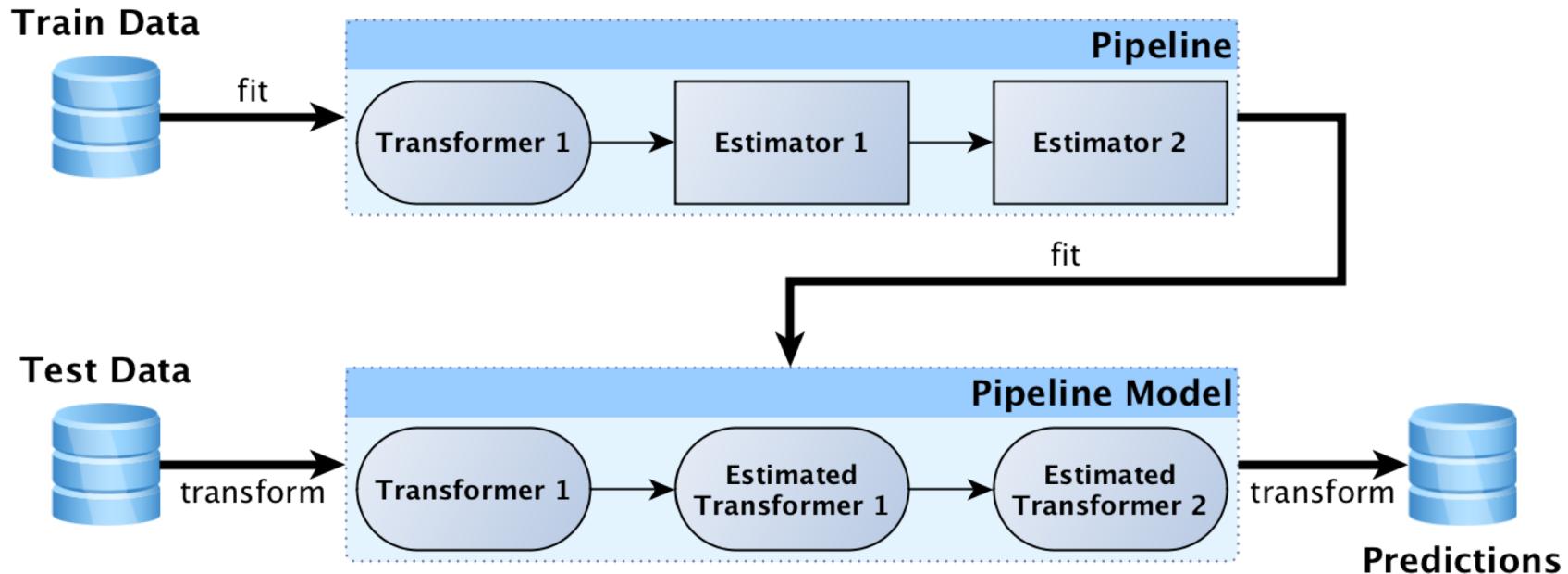
# Spark ML Pipelines



# Spark ML Pipelines



# Spark ML Pipelines



# Пример 1: Генерируем данные

```
val X = DenseMatrix.rand(100000, 3)
val y = X * DenseVector(0.5,-0.1,0.2)
val data = DenseMatrix.horzcat(X, y.asDenseMatrix.t)
```

```
val df = data(*, ::).iterator
    .map(x => (x(0),x(1),x(2),x(3)))
    .toSeq.toDF("x1","x2","x3","y")
```

```
df.show(1)
```

```
df: org.apache.spark.sql.DataFrame = [x1: double, x2: double ... 2 more fields]
+-----+-----+-----+-----+
|          x1|          x2|          x3|          y|
+-----+-----+-----+-----+
| 0.8015794300823613| 0.6690293703312364| 0.37975321925621097| 0.40983742185929917|
+-----+-----+-----+-----+
```

# Пример 1: Генерируем данные

```
val X = DenseMatrix.rand(100000, 3)
val y = X * DenseVector(0.5,-0.1,0.2)
val data = DenseMatrix.horzcat(X, y.asDenseMatrix.t)
```

```
val df = data(*, ::).iterator
    .map(x => (x(0),x(1),x(2),x(3)))
    .toSeq.toDF("x1","x2","x3","y")
```

```
df.show(1)
```

```
df: org.apache.spark.sql.DataFrame = [x1: double, x2: double ... 2 more fields]
+-----+-----+-----+-----+
|          x1|          x2|          x3|          y|
+-----+-----+-----+-----+
| 0.8015794300823613| 0.6690293703312364| 0.37975321925621097| 0.40983742185929917|
+-----+-----+-----+-----+
```

# Пример 1: Генерируем данные

```
val X = DenseMatrix.rand(100000, 3)
val y = X * DenseVector(0.5, -0.1, 0.2)
val data = DenseMatrix.horzcat(X, y.asDenseMatrix.t)
```

```
val df = data(*, ::).iterator
    .map(x => (x(0),x(1),x(2),x(3)))
    .toSeq.toDF("x1","x2","x3","y")
```

```
df.show(1)
```

```
df: org.apache.spark.sql.DataFrame = [x1: double, x2: double ... 2 more fields]
+-----+-----+-----+-----+
|      x1|      x2|      x3|      y|
+-----+-----+-----+-----+
| 0.8015794300823613| 0.6690293703312364| 0.37975321925621097| 0.40983742185929917|
+-----+-----+-----+-----+
```

# Пример 1: Генерируем данные

```
val X = DenseMatrix.rand(100000, 3)
val y = X * DenseVector(0.5,-0.1,0.2)
val data = DenseMatrix.horzcat(X, y.asDenseMatrix.t)
```

```
val df = data(*, ::).iterator
    .map(x => (x(0),x(1),x(2),x(3)))
    .toSeq.toDF("x1","x2","x3","y")
```

```
df.show(1)
```

```
df: org.apache.spark.sql.DataFrame = [x1: double, x2: double ... 2 more fields]
```

	x1	x2	x3	y
	10.8015794300823613	10.6690293703312364	10.379753219256210971	10.409837421859299171

# Пример 1: Тренируем модель

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
```

```
val pipeline = new Pipeline().setStages(Array(
    new VectorAssembler()
        .setInputCols(Array("x1", "x2", "x3"))
        .setOutputCol("features"),
    new LinearRegression().setLabelCol("y")
))

val model = pipeline.fit(df)

val w = model.stages.last.asInstanceOf[LinearRegressionModel].coefficients

pipeline: org.apache.spark.ml.Pipeline = pipeline_142f75763be2
model: org.apache.spark.ml.PipelineModel = pipeline_142f75763be2
w: org.apache.spark.ml.linalg.Vector = [0.5000000000000011,-0.1000000000000009,0.2000000000000065]
```

# Пример 1: Тренируем модель

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
```

```
val pipeline = new Pipeline().setStages(Array(
    new VectorAssembler()
        .setInputCols(Array("x1", "x2", "x3"))
        .setOutputCol("features"),
    new LinearRegression().setLabelCol("y")
))

val model = pipeline.fit(df)

val w = model.stages.last.asInstanceOf[LinearRegressionModel].coefficients

pipeline: org.apache.spark.ml.Pipeline = pipeline_142f75763be2
model: org.apache.spark.ml.PipelineModel = pipeline_142f75763be2
w: org.apache.spark.ml.linalg.Vector = [0.5000000000000011,-0.1000000000000009,0.2000000000000065]
```

# Пример 1: Тренируем модель

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
```

```
val pipeline = new Pipeline().setStages(Array(
    new VectorAssembler()
        .setInputCols(Array("x1", "x2", "x3"))
        .setOutputCol("features"),
    new LinearRegression().setLabelCol("y")
))
```

```
val model = pipeline.fit(df)
```

```
val w = model.stages.last.asInstanceOf[LinearRegressionModel].coefficients
```

```
pipeline: org.apache.spark.ml.Pipeline = pipeline_142f75763be2
```

```
model: org.apache.spark.ml.PipelineModel = pipeline_142f75763be2
```

```
w: org.apache.spark.ml.linalg.Vector = [0.5000000000000011,-0.1000000000000009,0.2000000000000065]
```

# Пример 1: Тренируем модель

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
```

```
val pipeline = new Pipeline().setStages(Array(
    new VectorAssembler()
        .setInputCols(Array("x1", "x2", "x3"))
        .setOutputCol("features"),
    new LinearRegression().setLabelCol("y")
))
```

```
val model = pipeline.fit(df)
```

```
val w = model.stages.last.asInstanceOf[LinearRegressionModel].coefficients
```

```
pipeline: org.apache.spark.ml.Pipeline = pipeline_142f75763be2
```

```
model: org.apache.spark.ml.PipelineModel = pipeline_142f75763be2
```

```
w: org.apache.spark.ml.linalg.Vector = [0.5000000000000011,-0.1000000000000009,0.2000000000000065]
```

# Пример 1: Тренируем модель

```
import org.apache.spark.ml.Pipeline
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.regression.{LinearRegression, LinearRegressionModel}
```

```
val pipeline = new Pipeline().setStages(Array(
    new VectorAssembler()
        .setInputCols(Array("x1", "x2", "x3"))
        .setOutputCol("features"),
    new LinearRegression().setLabelCol("y")
))

val model = pipeline.fit(df)

val w = model.stages.last.asInstanceOf[LinearRegressionModel].coefficients

pipeline: org.apache.spark.ml.Pipeline = pipeline_142f75763be2
model: org.apache.spark.ml.PipelineModel = pipeline_142f75763be2
w: org.apache.spark.ml.linalg.Vector = [0.5000000000000011,-0.1000000000000009,0.2000000000000065]
```

# Пример 1: Вычисление прогноза

```
val predictions = model.transform(df)
predictions.show(10)

predictions: org.apache.spark.sql.DataFrame = [x1: double, x2: double ... 4 more fields]
+-----+-----+-----+-----+-----+-----+
|          x1|       x2|       x3|       y| features| prediction|
+-----+-----+-----+-----+-----+-----+
| 0.8015794300823613| 0.6690293703312364| 0.37975321925621097| 0.40983742185929917|[0.80157943008236...| 0.40983742185929941
| 0.8541056821815281| 0.8265355819625297| 0.45544701134477505| 0.43548868516346606|[0.85410568218152...| 0.435488685163466341
| 0.05301363103882851| 0.4415241954196951| 0.43366245184484176| 0.06908688634641308|[0.05301363103882...| 0.069086886346412511
| 0.49020733645910551| 0.5775215033969785| 0.11712733852871482| 0.21077698559559788|[0.49020733645910...| 0.210776985595597541
| 0.300817560829979461| 0.3602281459209993| 0.8794421521153877| 0.29027439624596735|[0.30081756082997...| 0.29027439624596731
| 0.5429499131639877| 0.039550525928484515| 0.0871010994394894| 0.28494012387704326|[0.54294991316398...| 0.284940123877043041
| 0.91340567166830661| 0.10454074352127263| 0.4874279419513512| 0.5437343498722963|[0.91340567166830...| 0.54373434987229661
| 0.2218761027760131| 0.8580765943456652| 0.9239292043604319| 0.20991623282552635|[0.22187610277601...| 0.209916232825526241
| 0.78617500664070561| 0.3583022215080267| 0.7828116759888151| 0.5138196163673132|[0.78617500664070...| 0.51381961636731371
| 0.72833731023232181| 0.8282171128884972| 0.4445877207042108| 0.37026448796815337|[0.72833731023232...| 0.370264487968153531
+-----+-----+-----+-----+-----+-----+
```

# Пример 1: Вычисление прогноза

```
val predictions = model.transform(df)
predictions.show(10)
```

```
predictions: org.apache.spark.sql.DataFrame = [x1: double, x2: double ... 4 more fields]
+-----+-----+-----+-----+-----+-----+
|          x1|       x2|       x3|       y| features| prediction|
+-----+-----+-----+-----+-----+-----+
| 0.8015794300823613| 0.6690293703312364| 0.37975321925621097| 0.40983742185929917|[0.80157943008236...| 0.40983742185929941
| 0.8541056821815281| 0.8265355819625297| 0.45544701134477505| 0.43548868516346606|[0.85410568218152...| 0.435488685163466341
| 0.05301363103882851| 0.4415241954196951| 0.43366245184484176| 0.06908688634641308|[0.05301363103882...| 0.069086886346412511
| 0.49020733645910551| 0.5775215033969785| 0.11712733852871482| 0.21077698559559788|[0.49020733645910...| 0.210776985595597541
| 0.300817560829979461| 0.3602281459209993| 0.8794421521153877| 0.29027439624596735|[0.30081756082997...| 0.29027439624596731
| 0.5429499131639877| 0.039550525928484515| 0.0871010994394894| 0.28494012387704326|[0.54294991316398...| 0.284940123877043041
| 0.91340567166830661| 0.10454074352127263| 0.4874279419513512| 0.5437343498722963|[0.91340567166830...| 0.54373434987229661
| 0.2218761027760131| 0.8580765943456652| 0.9239292043604319| 0.20991623282552635|[0.22187610277601...| 0.209916232825526241
| 0.78617500664070561| 0.3583022215080267| 0.7828116759888151| 0.5138196163673132|[0.78617500664070...| 0.51381961636731371
| 0.72833731023232181| 0.8282171128884972| 0.4445877207042108| 0.37026448796815337|[0.72833731023232...| 0.370264487968153531
+-----+-----+-----+-----+-----+-----+
```

# SQL тип VectorUDT

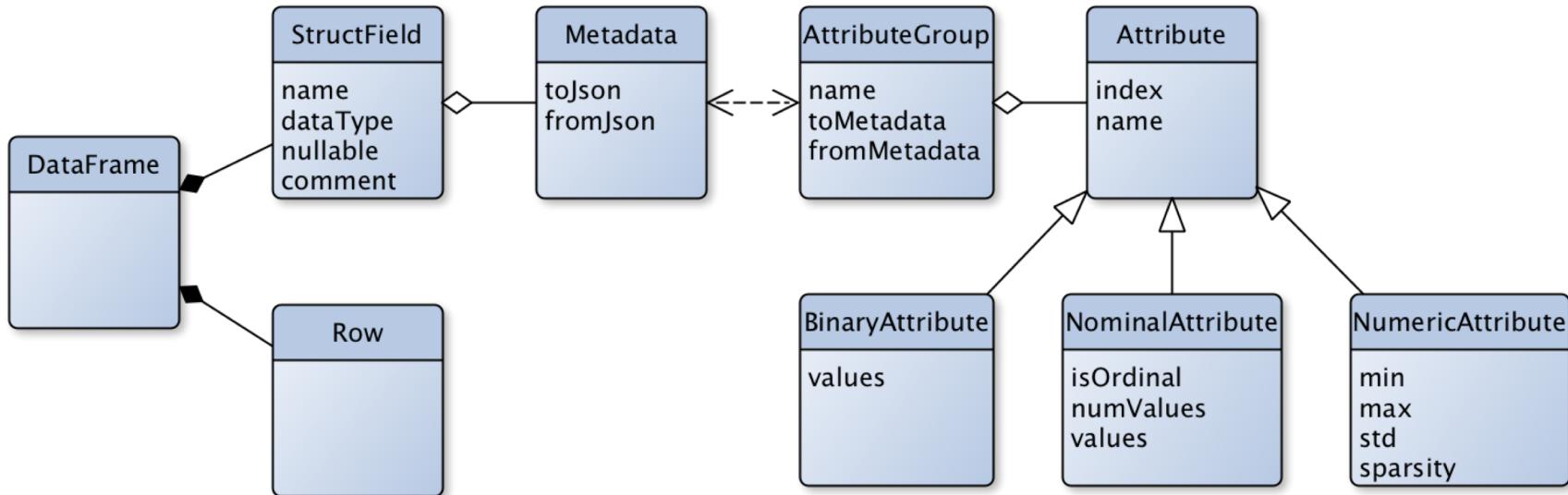
```
| predictions.printSchema
```

```
root
|--- x1: double (nullable = false)
|--- x2: double (nullable = false)
|--- x3: double (nullable = false)
|--- y: double (nullable = false)
|--- features: vector (nullable = true) ←
|--- prediction: double (nullable = false)
```

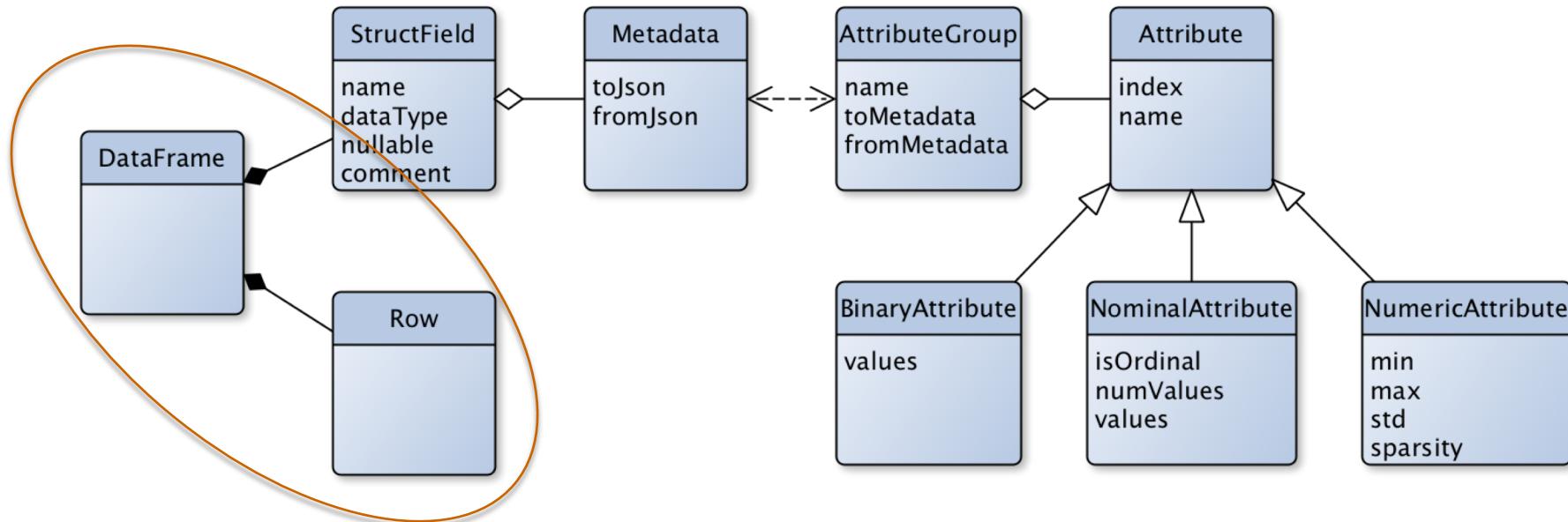
# org.apache.spark.ml.linalg

- Vector
  - DenseVector
  - SparseVector
- Matrix
  - DenseMatrix
  - SparseMatrix (CSC)
- Хранение в DataFrame
- compressed
- *Базовая арифметика поверх BLAS*
- *asBreeze/fromBreeze*

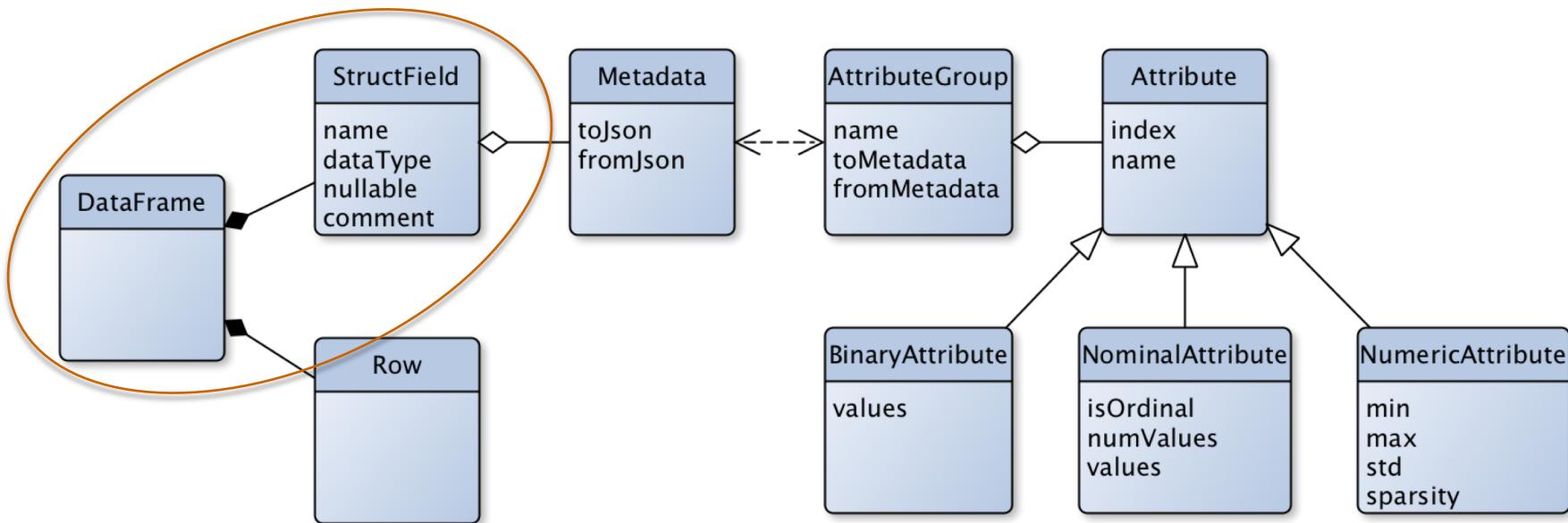
# Field Metadata and Attributes



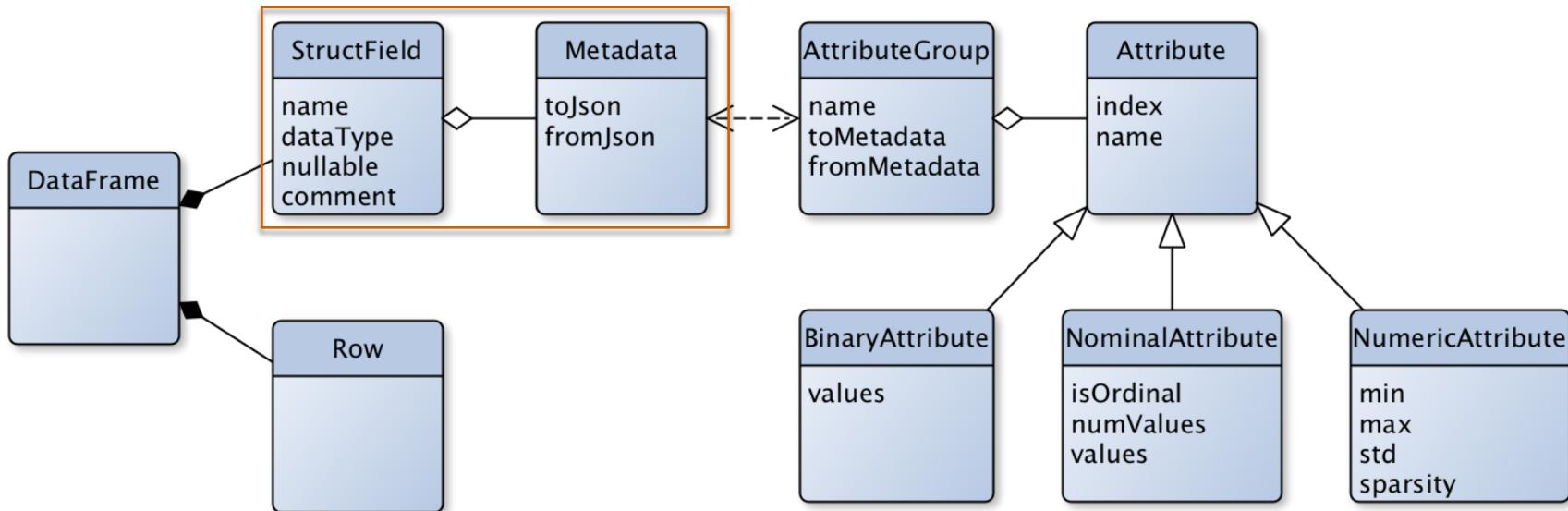
# Field Metadata and Attributes



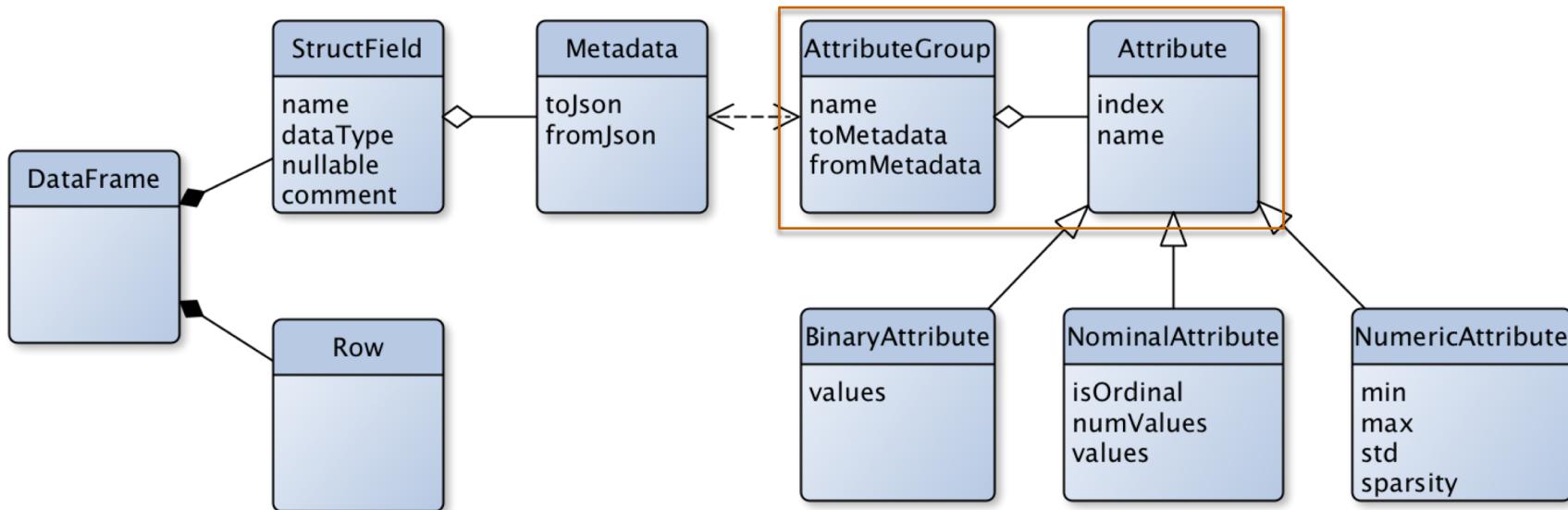
# Field Metadata and Attributes



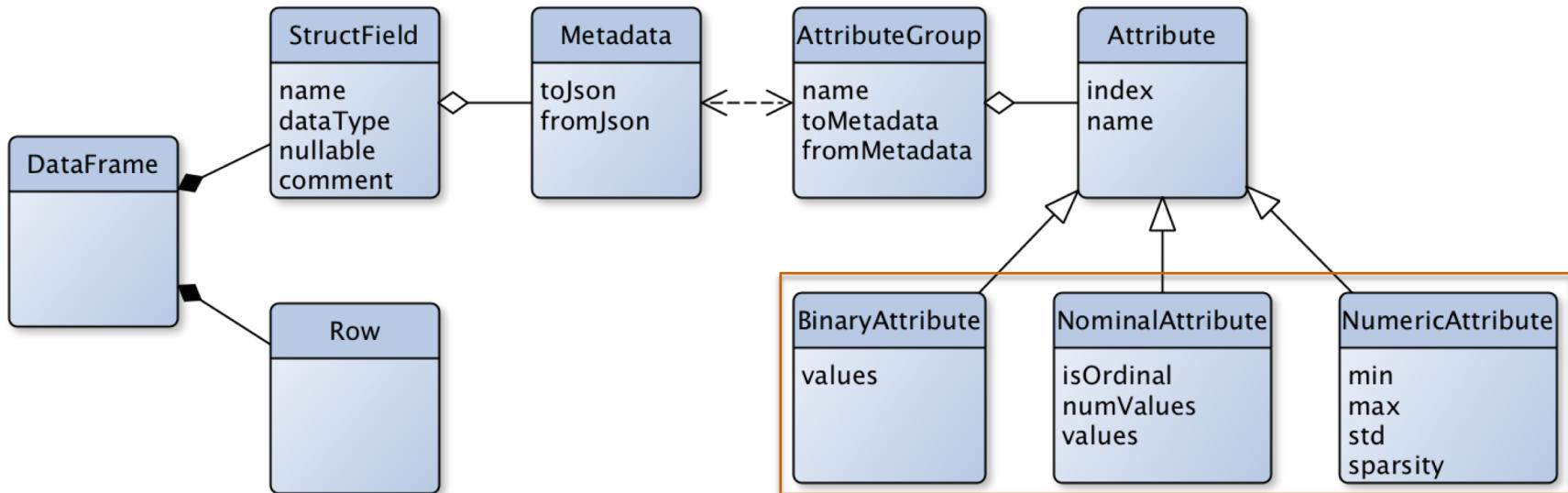
# Field Metadata and Attributes



# Field Metadata and Attributes



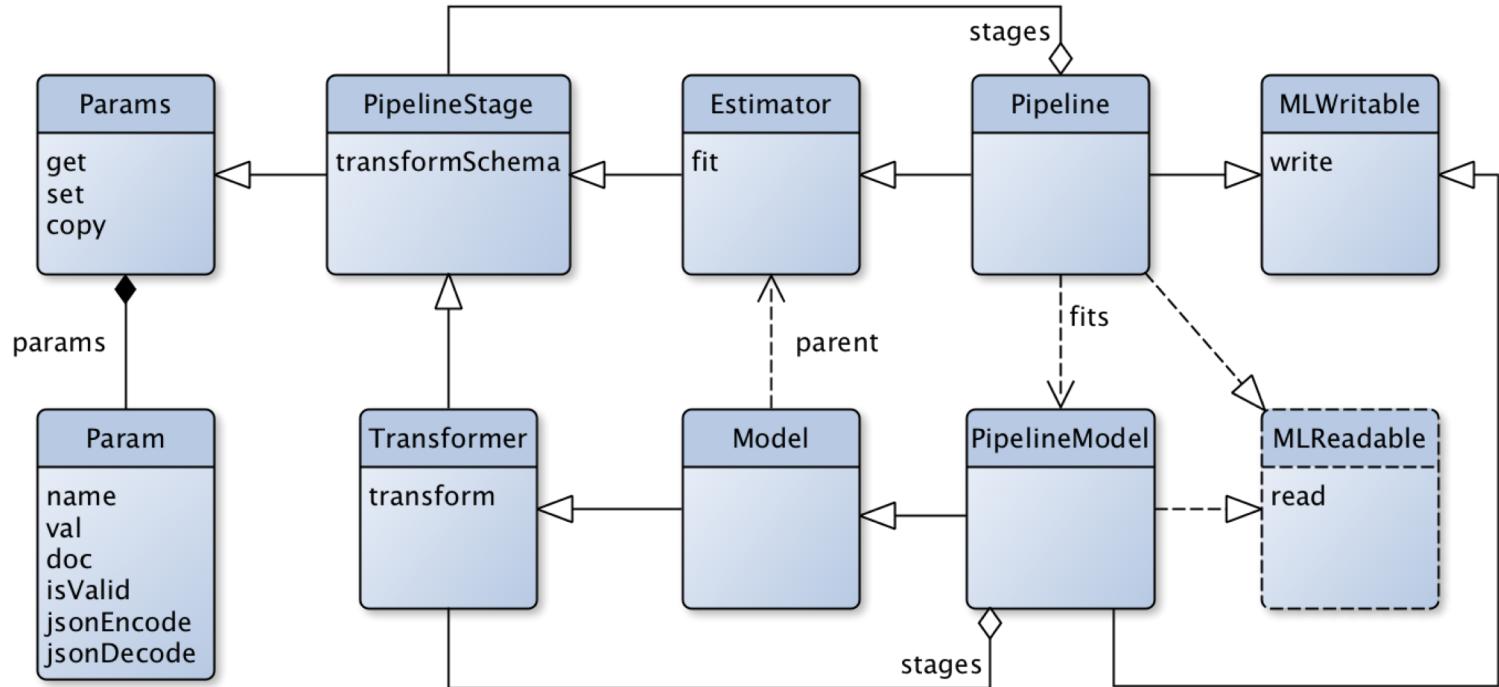
# Field Metadata and Attributes



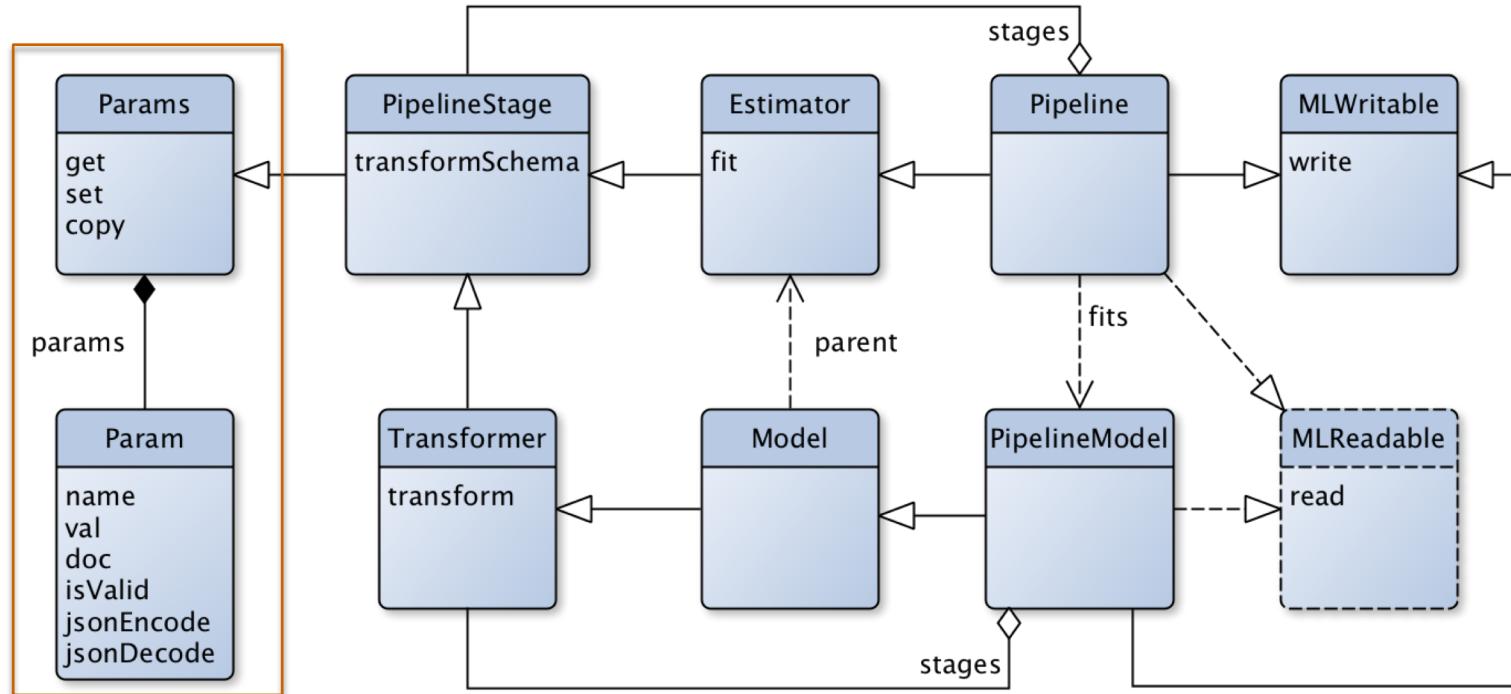
# Извлечение атрибутов

```
import org.apache.spark.ml.attribute.AttributeGroup  
  
AttributeGroup.fromStructField(predictions.schema("features"))  
    .attributes.get.foreach(println)  
  
import org.apache.spark.ml.attribute.AttributeGroup  
{"type": "numeric", "idx": 0, "name": "x1"}  
{"type": "numeric", "idx": 1, "name": "x2"}  
{"type": "numeric", "idx": 2, "name": "x3"}
```

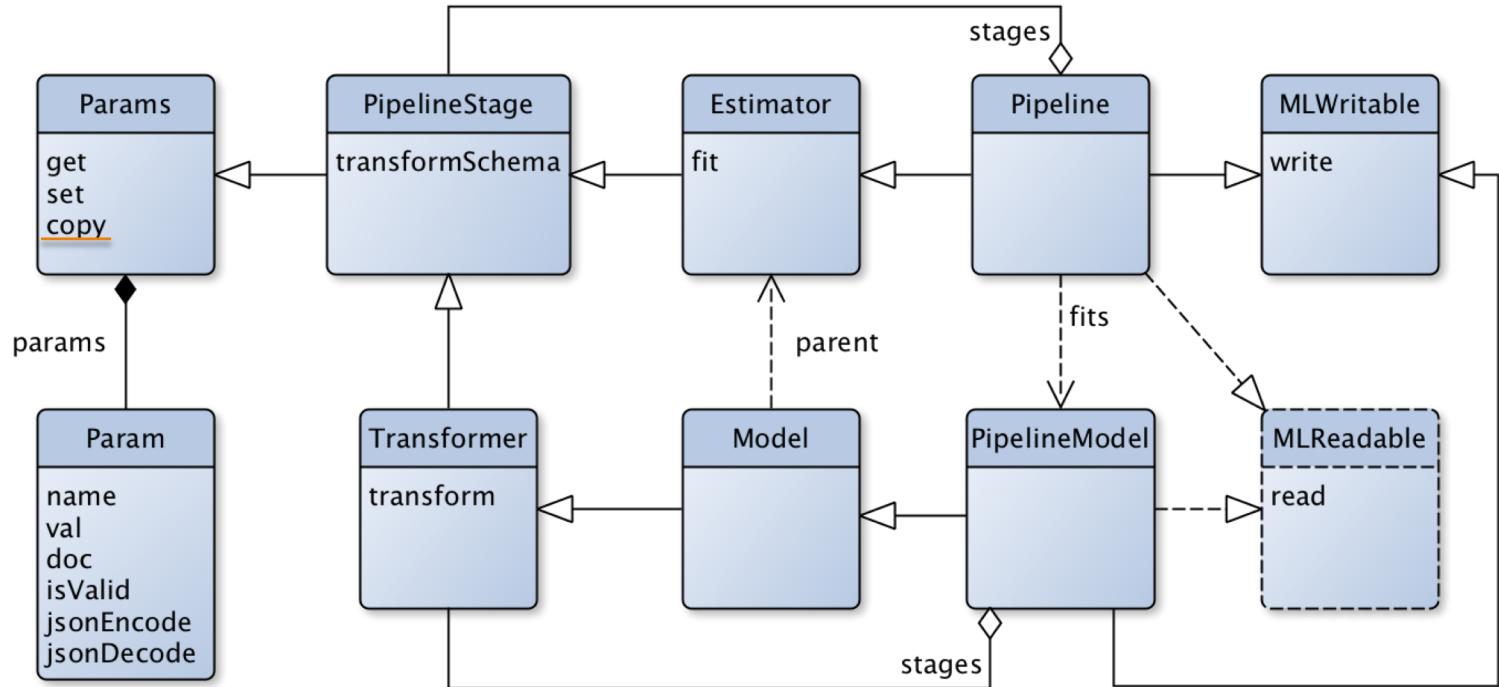
# Spark ML Core



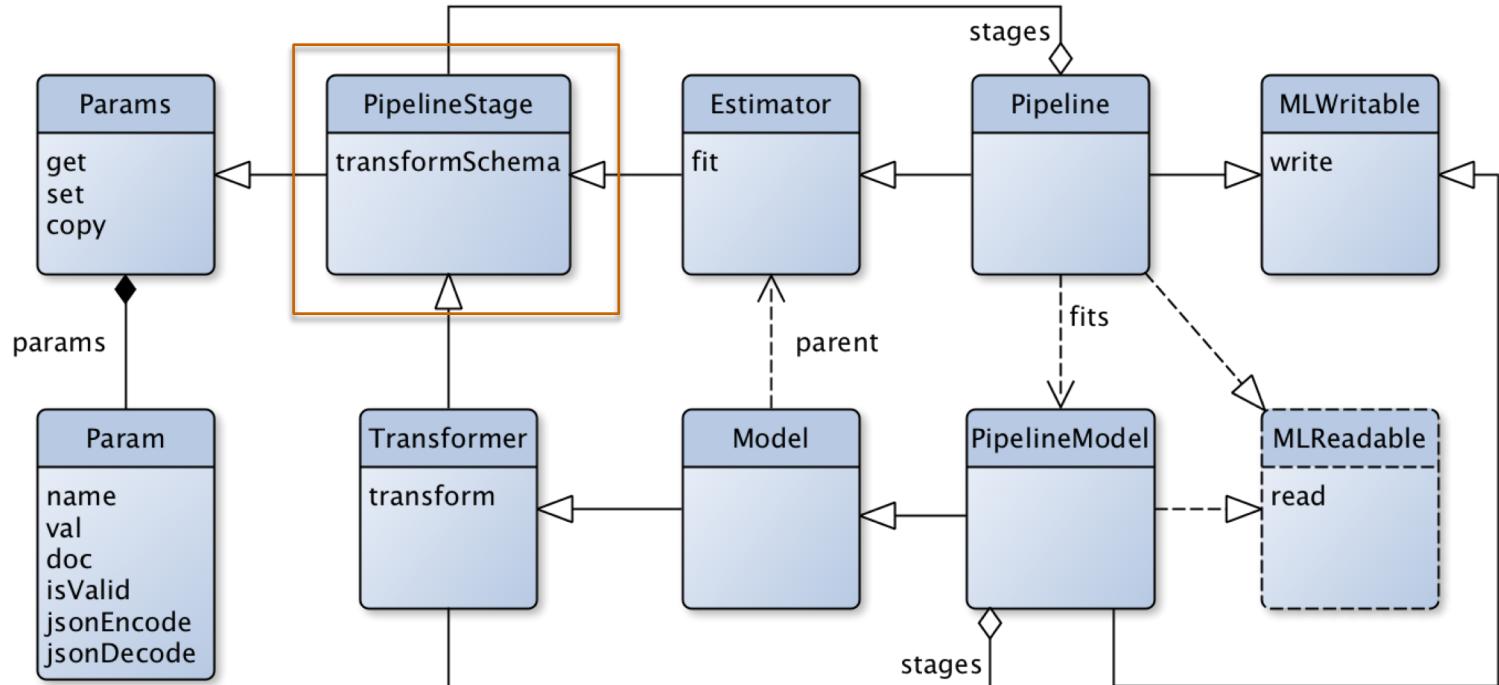
# Spark ML Core



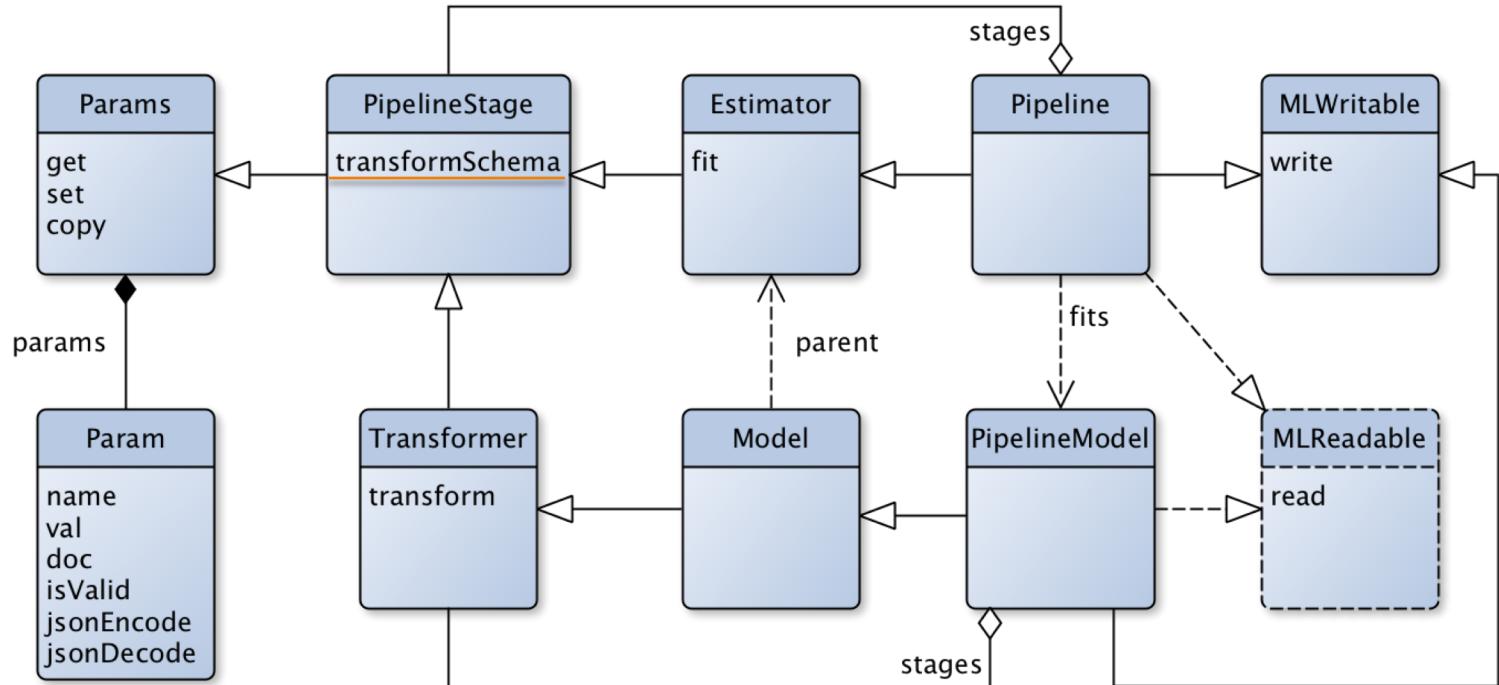
# Spark ML Core



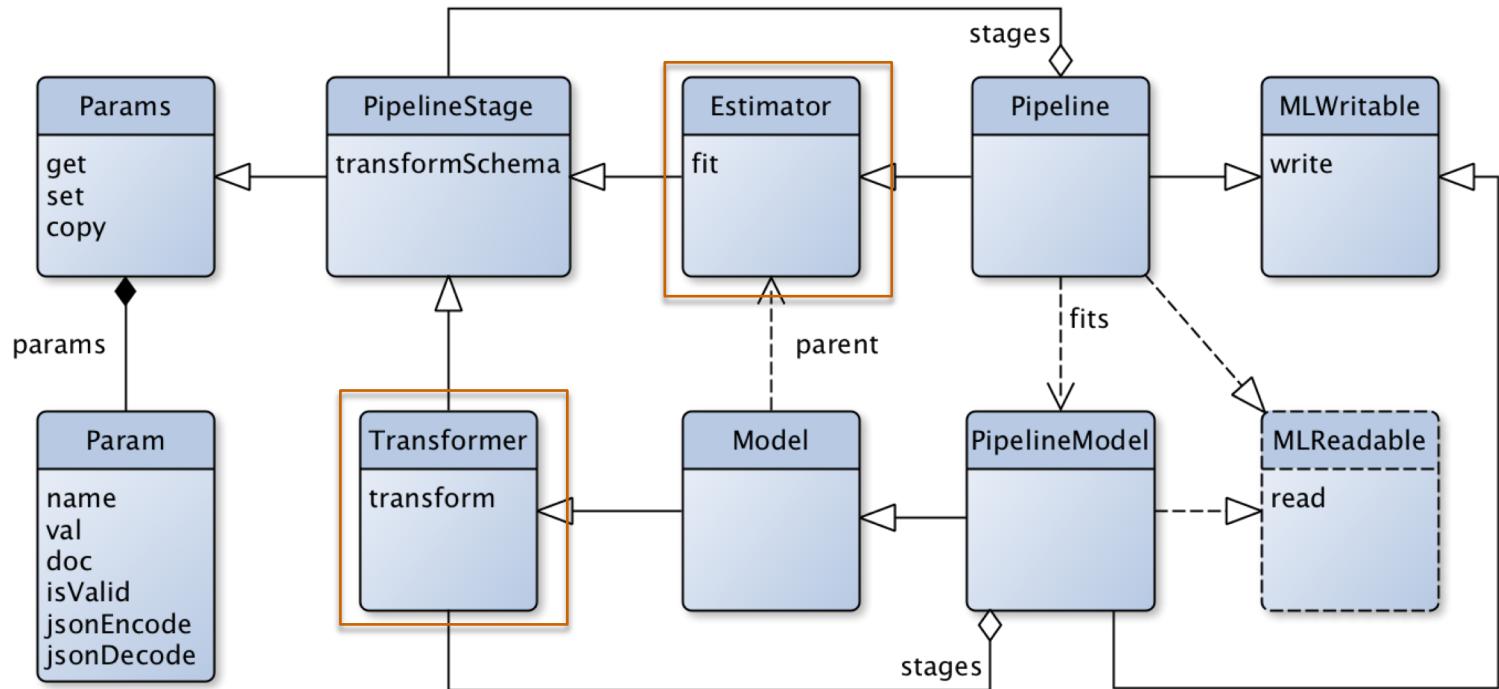
# Spark ML Core



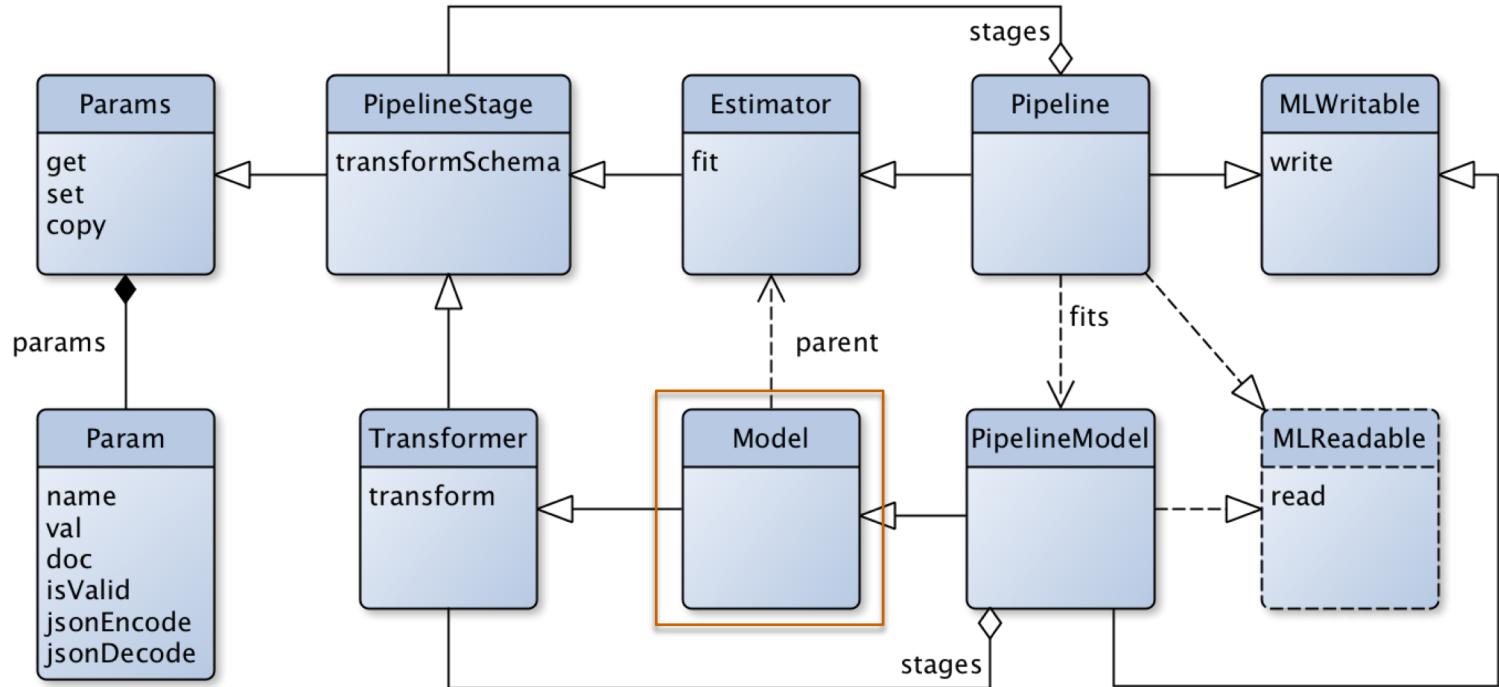
# Spark ML Core



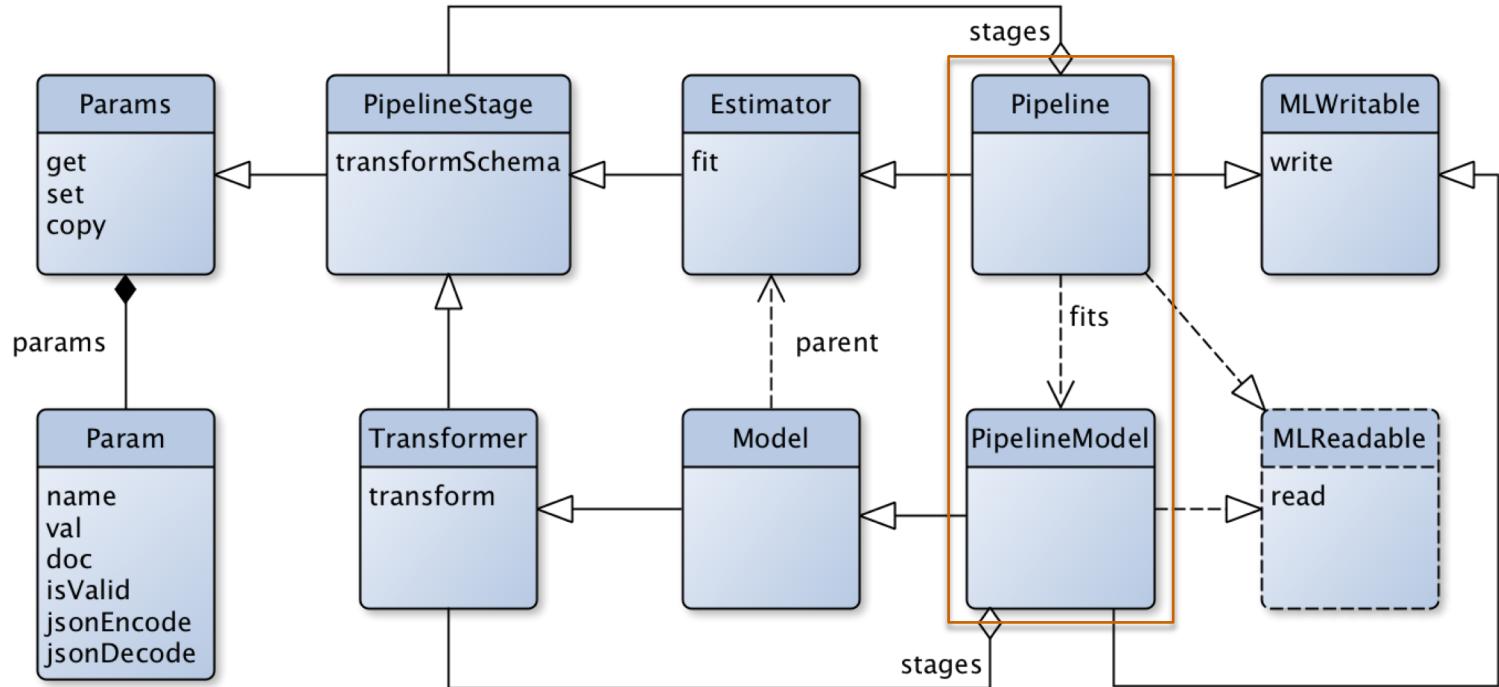
# Spark ML Core



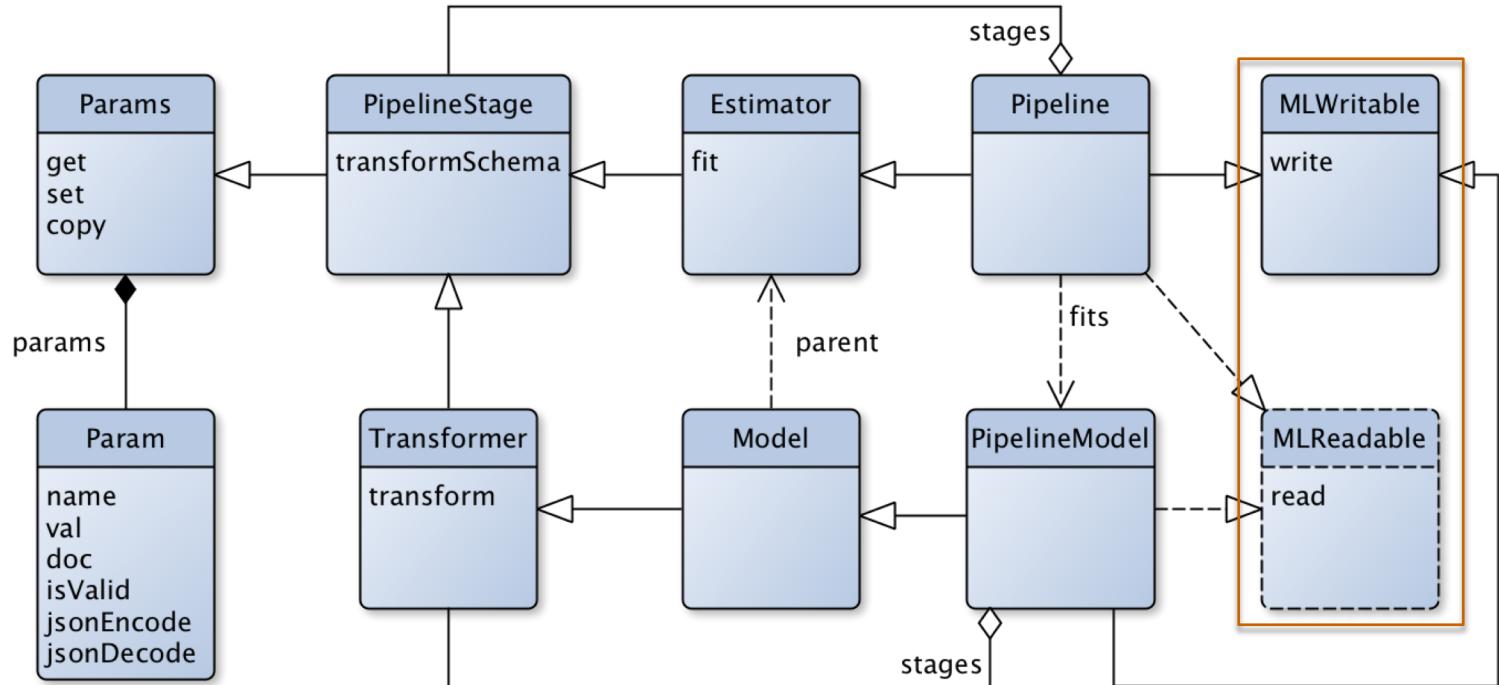
# Spark ML Core



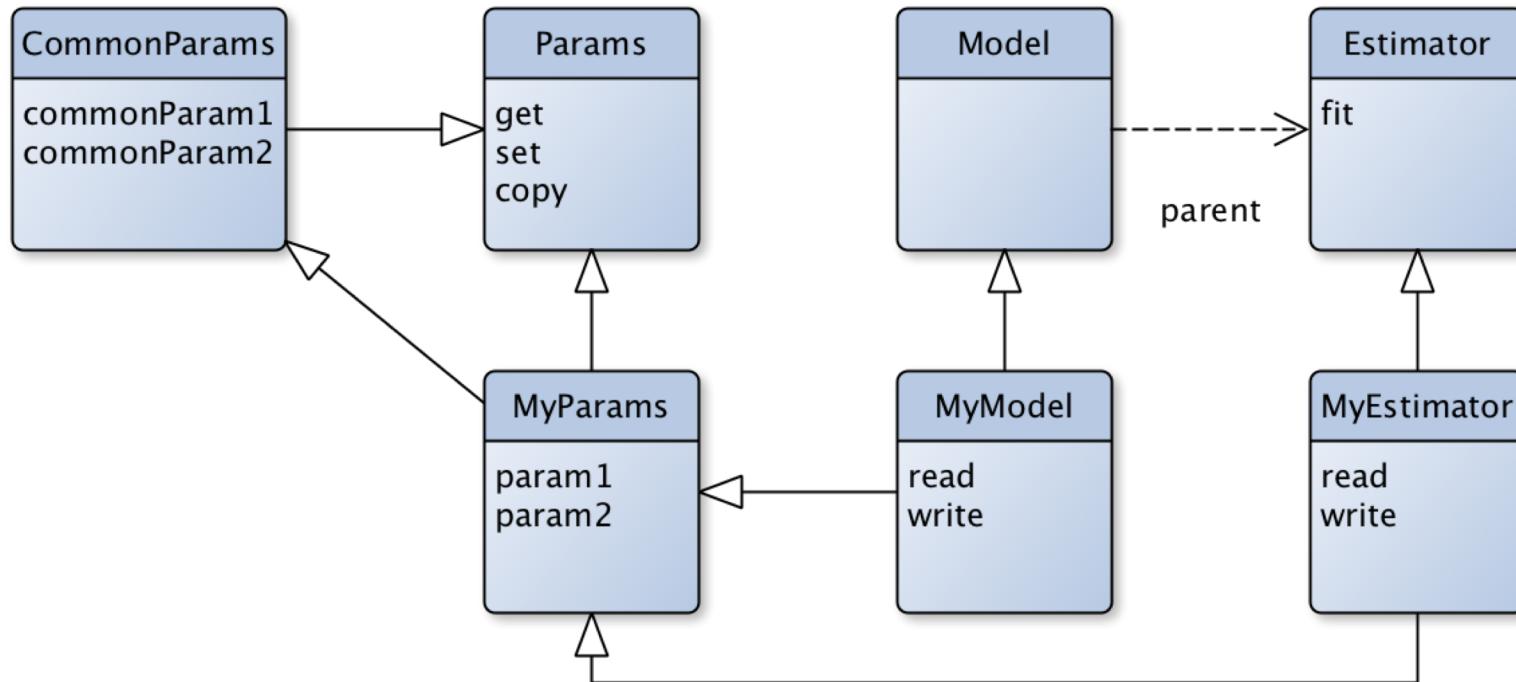
# Spark ML Core



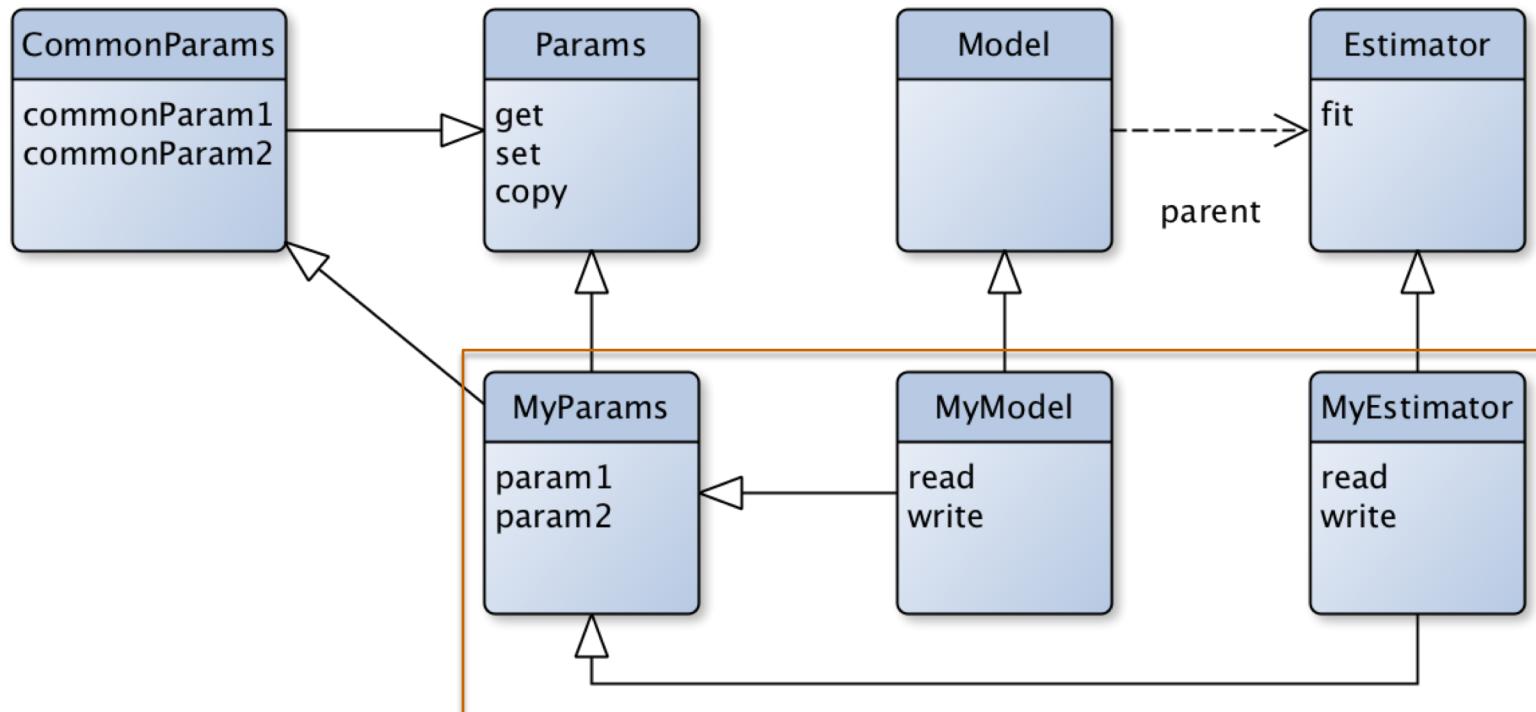
# Spark ML Core



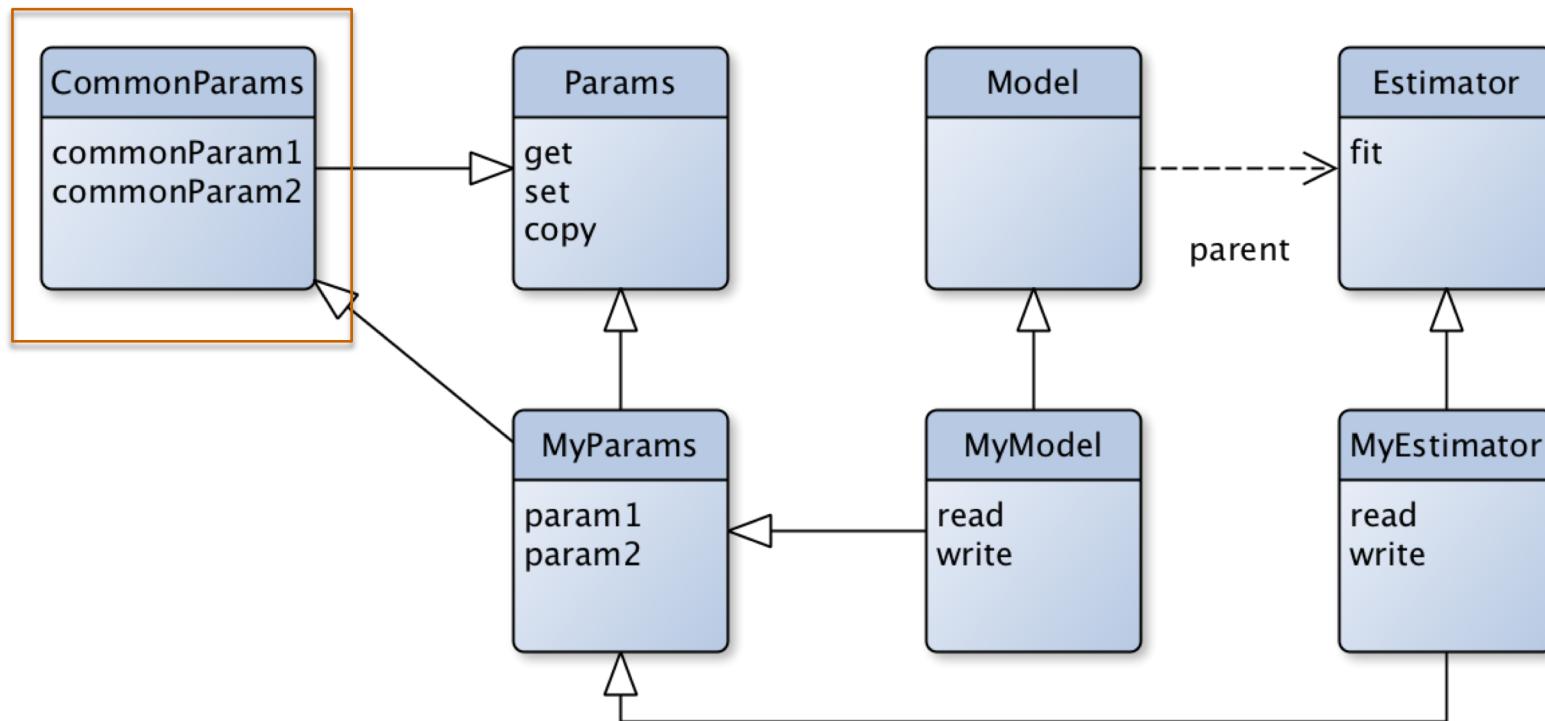
# “My Spark ML Model”



# “My Spark ML Model”



# “My Spark ML Model”



# ДЗ: Линейная регрессия Breeze + Spark ML

$$X \in \Re^{N \times d} \quad \vec{y} \in \Re^N$$

$$\vec{\varepsilon} = X \bullet \vec{w} + \vec{1} * b - \vec{y}$$

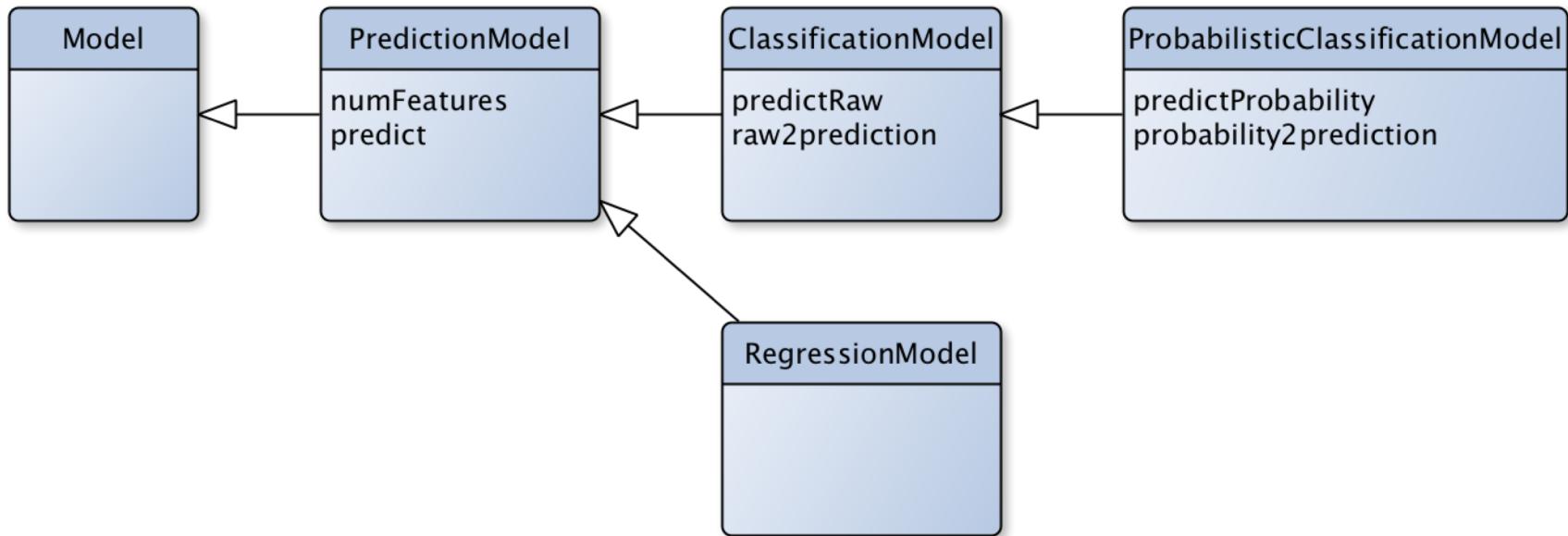
$$\vec{w}_{t+1} = \vec{w}_t - \frac{\lambda}{N} (\vec{\varepsilon}^T \bullet X)^T$$

$$b_{t+1} = b_t - \frac{\lambda}{N} \vec{1} \bullet \vec{\varepsilon}$$

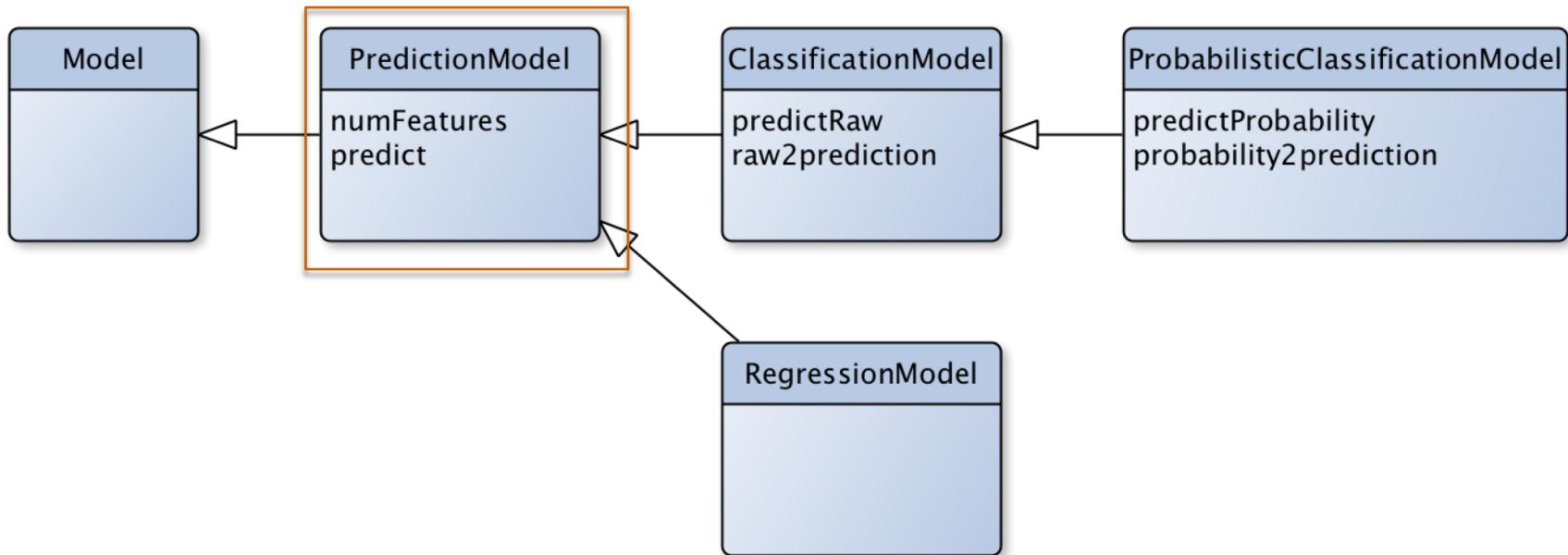
- Случайная матрица  $X$  100000 на 3
- «Скрытая модель» (1.5,0.3,-0.7)
- Задача: распределенным градиентным спуском раскрыть модель
- Задача\*: добавить минибатч с матричной арифметикой

<https://github.com/scalanlp/breeze>

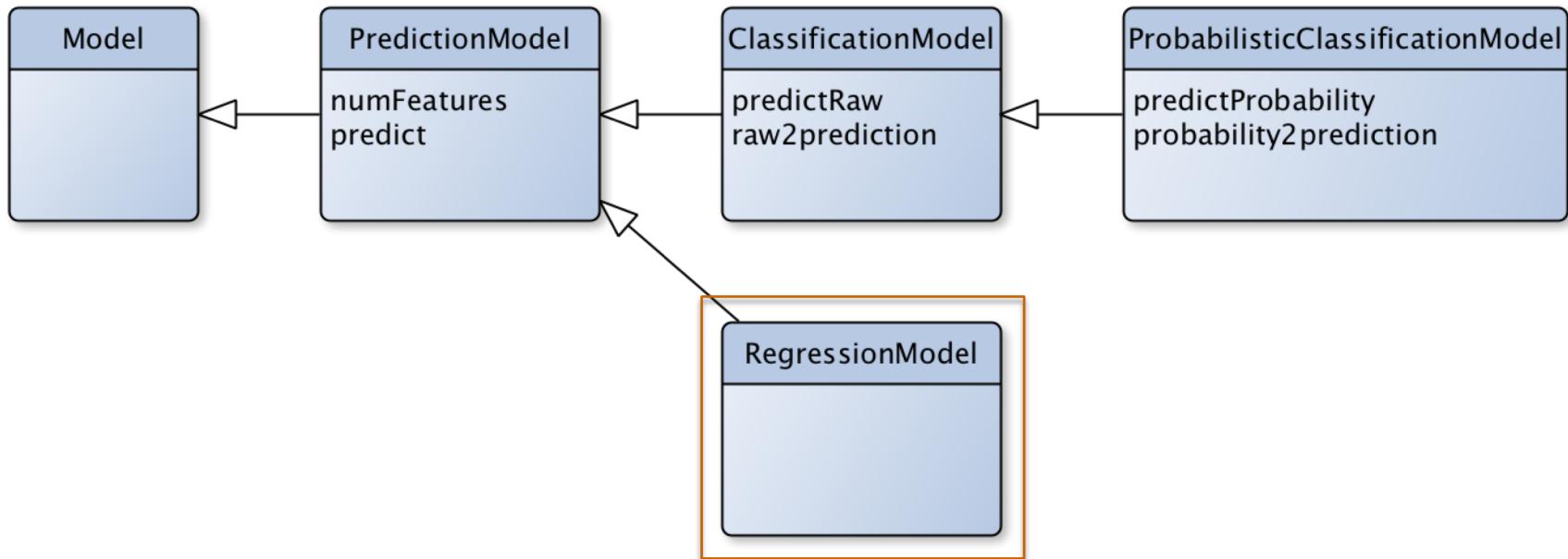
# Prediction Model



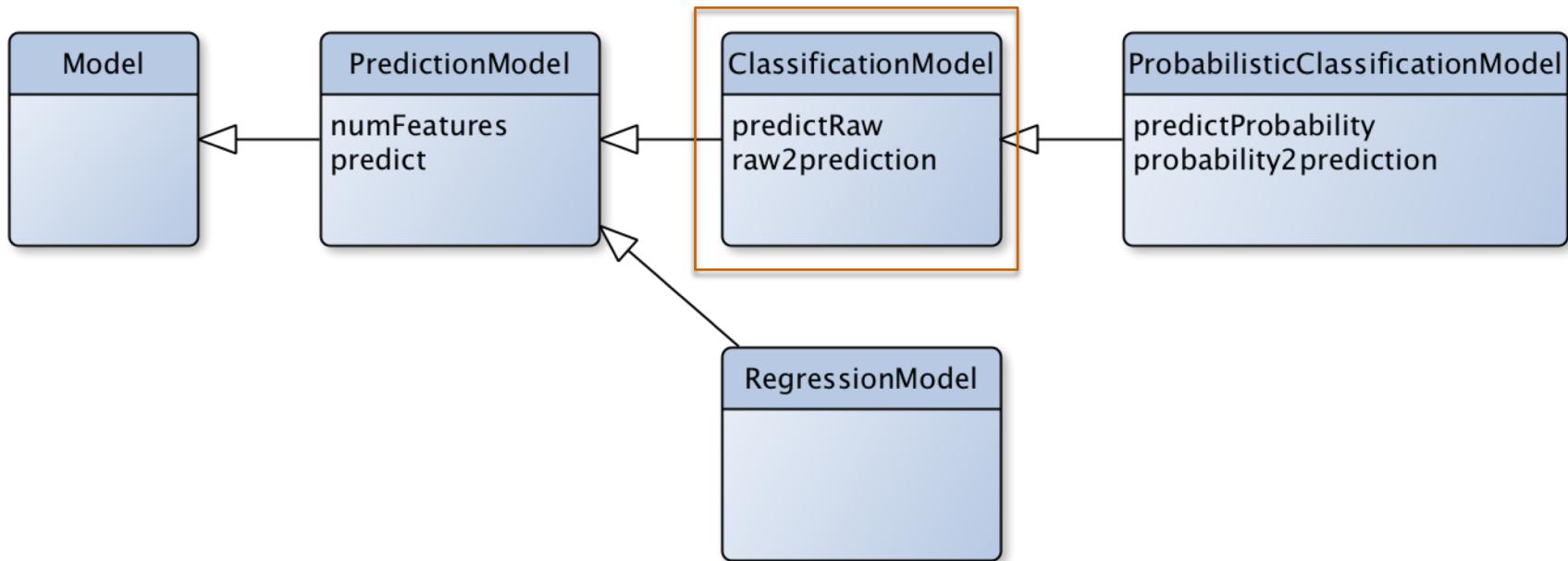
# Prediction Model



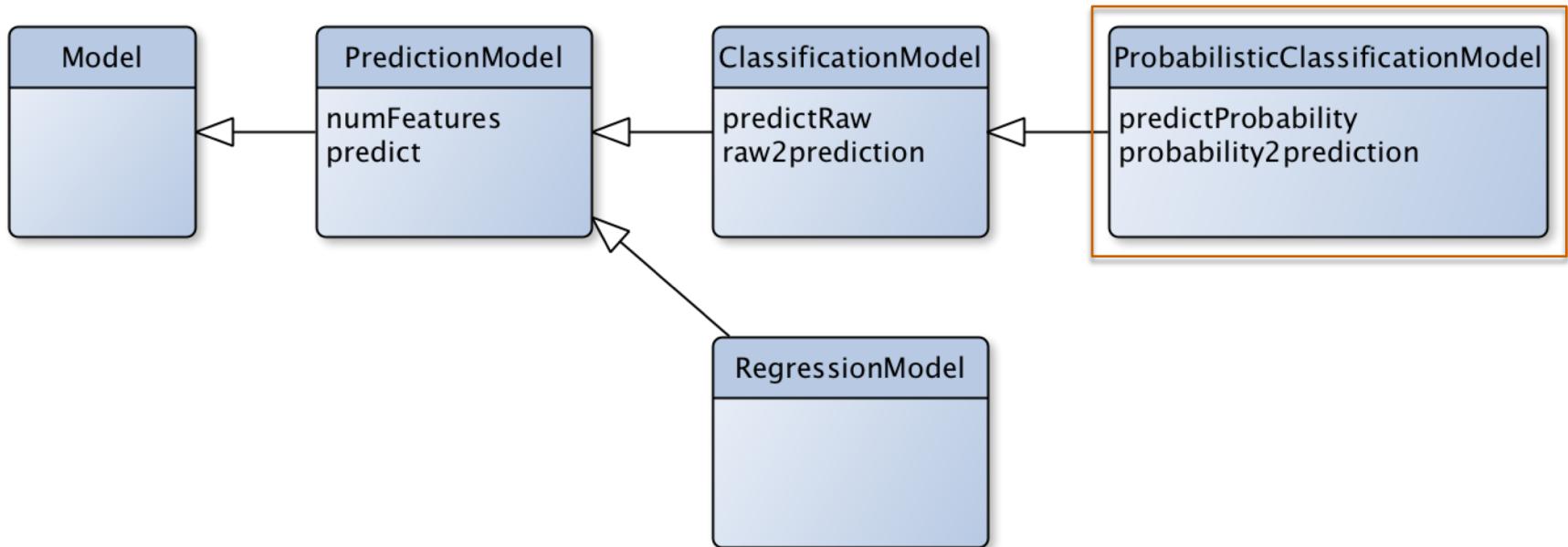
# Prediction Model



# Prediction Model



# Prediction Model



# Возможности Spark MLlib

- ETL
  - SQLTransformer
- Numerization
  - OneHotEncoder
  - StringIndexer
- Vectorization
  - VectorAssembler
  - FeatureHasher
- Feature Normalization
  - MaxAbsScaler
  - MinMaxScaler
  - Normalizer
  - QuantileDiscretizer
  - StandardScaler
- Missing values
  - Imputer

# Возможности Spark MLlib

- Feature Engineering
  - DCT
  - ElementwiseProduct
  - Interaction
  - VectorIndexer
  - PolynomialExpansion
- Feature Selection
  - ChiSqSelector
- Dimension reduction
  - PCA
  - MinHashLSHModel
  - BucketedRandomProjectionLSH

# Возможности Spark MLlib

- Texts extraction
  - Tokenizer
  - RegexTokenizer
  - Ngram
  - StopWordsRemover
- Texts vecotization
  - CountVectorizer
  - HashingTF
  - IDF
- Text embedding
  - Word2Vec
- Clustering
  - LDA
  - KMeans/BisectingKMeans
  - GaussianMixture

# Возможности Spark MLlib

## Regression

-  AFTSurvivalRegression.scala
-  DecisionTreeRegressor.scala
-  GBTRegressor.scala
-  GeneralizedLinearRegression.scala
-  IsotonicRegression.scala
-  LinearRegression.scala
-  RandomForestRegressor.scala

## Classification

- 
-  DecisionTreeClassifier.scala
  -  GBTClassifier.scala
  -  LinearSVC.scala
  -  LogisticRegression.scala
  -  MultilayerPerceptronClassifier.scala
  -  NaiveBayes.scala
  -  OneVsRest.scala
  -  ProbabilisticClassifier.scala
  -  RandomForestClassifier.scala

# Возможности Spark MLlib

- Recommendations
  - ALS
  - FP-Growth
- Evaluation
  - BinaryClassificationEvaluator
  - ClusteringEvaluator
  - MulticlassClassificationEvaluator  
or
  - RegressionEvaluator
- Tuning
  - ParamGridBuilder
  - CrossValidator