Big Data-Science

in Scala

Anastasia Lieva
Data Scientist
 @lievAnastazia

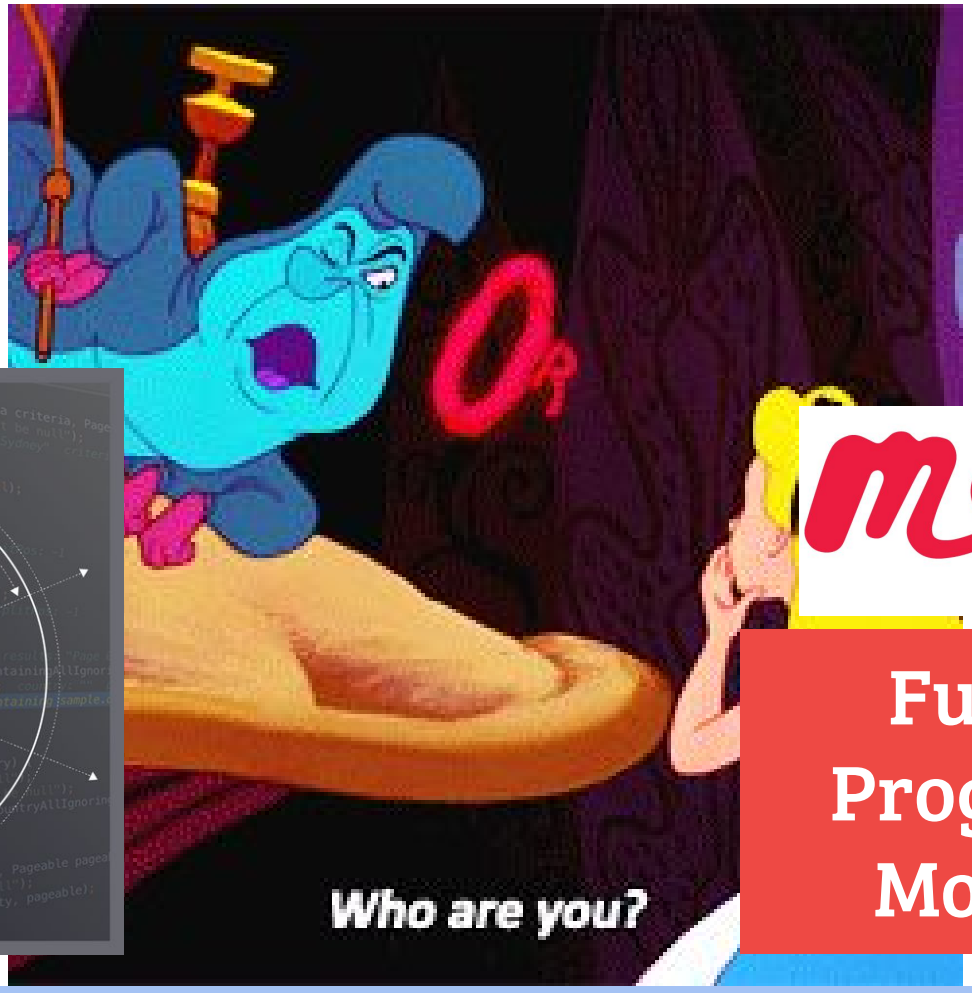

Who are you?

TABMO

CREATIVE MOBILE DSP

TABMO

CREATIVE MOBILE DSP



Meetup

Functional Programming Montpellier

План .

1. **Почему Scala ?**
2. **Обзор библиотек Data-Science на Scala**
3. **Что за проблему решаем сегодня?**
4. **Замарать руки в коде**
5. **Подвести итоги и выбрать**

подходящую нам библиотеку!

План.

1. **Почему Scala ?**
2. **Обзор библиотек Data-Science на Scala**
3. **Что за проблему решаем сегодня?**
4. **Замарать руки в коде**
5. **Подвести итоги и выбрать**

подходящую нам библиотеку!

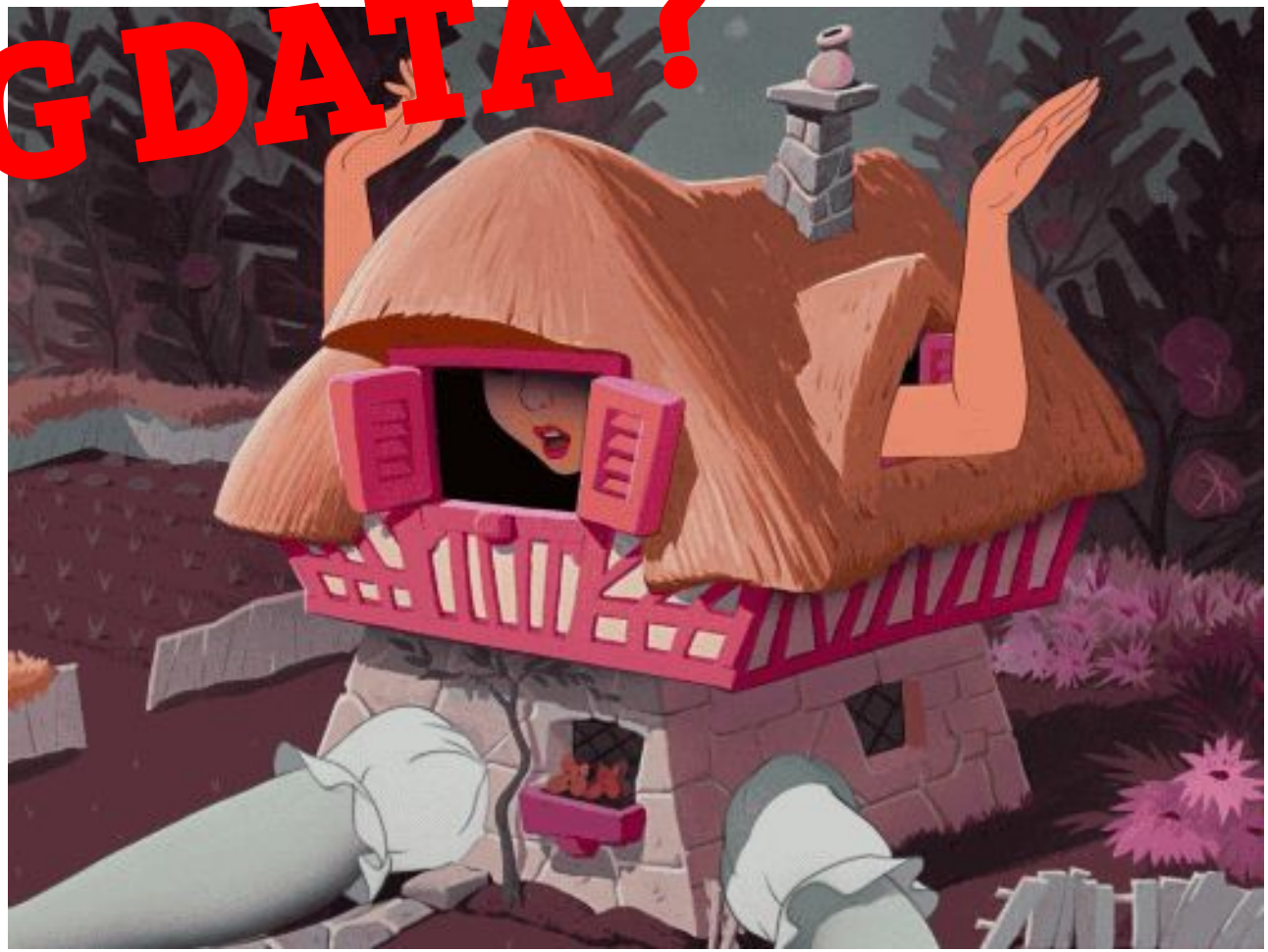
Сырые данные: 200 000 запросов в секунду

8 Терабайт в день

Технологии RTB

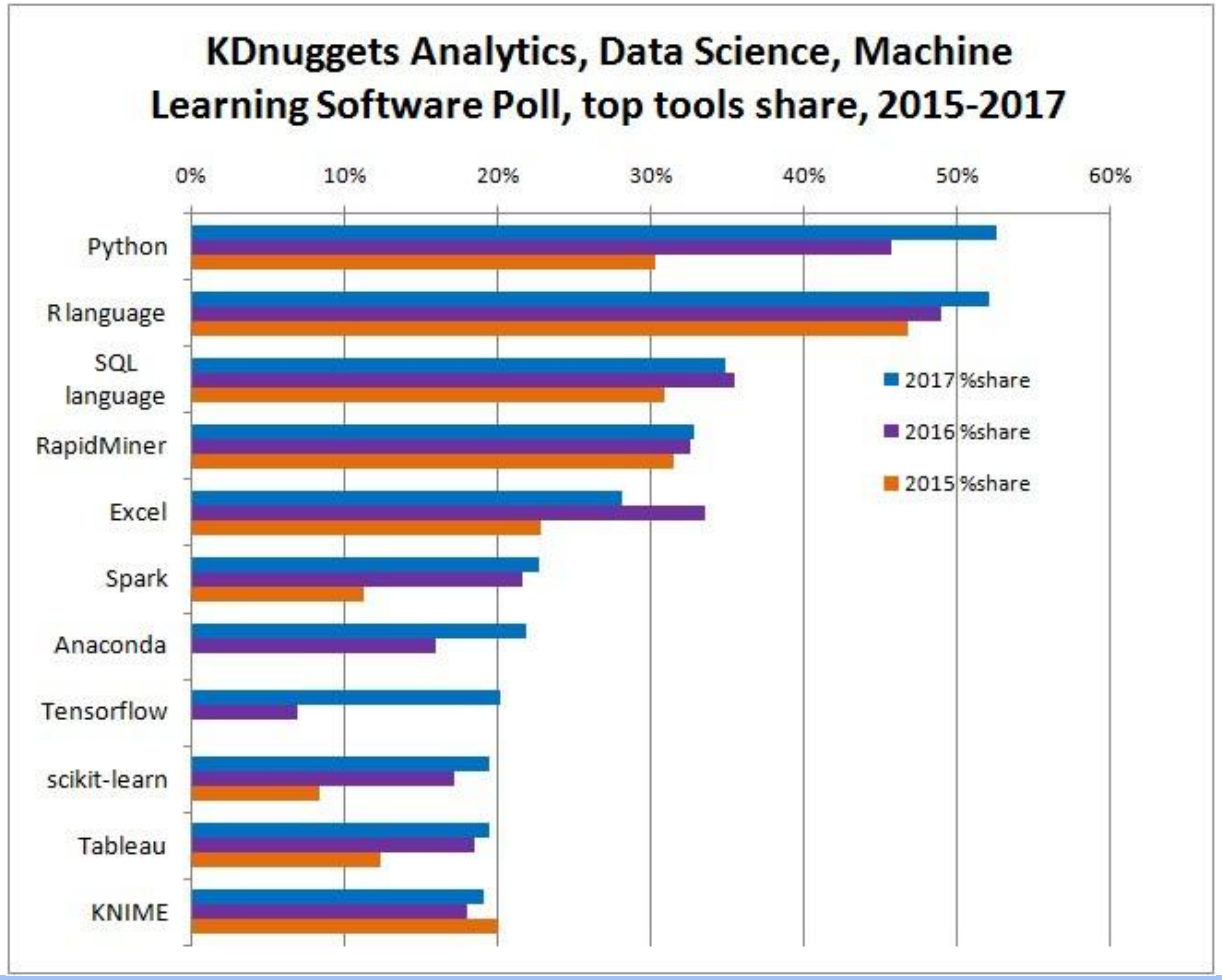
Сырые данные: 200 000 запросов в секунду

8 Терабайт в день



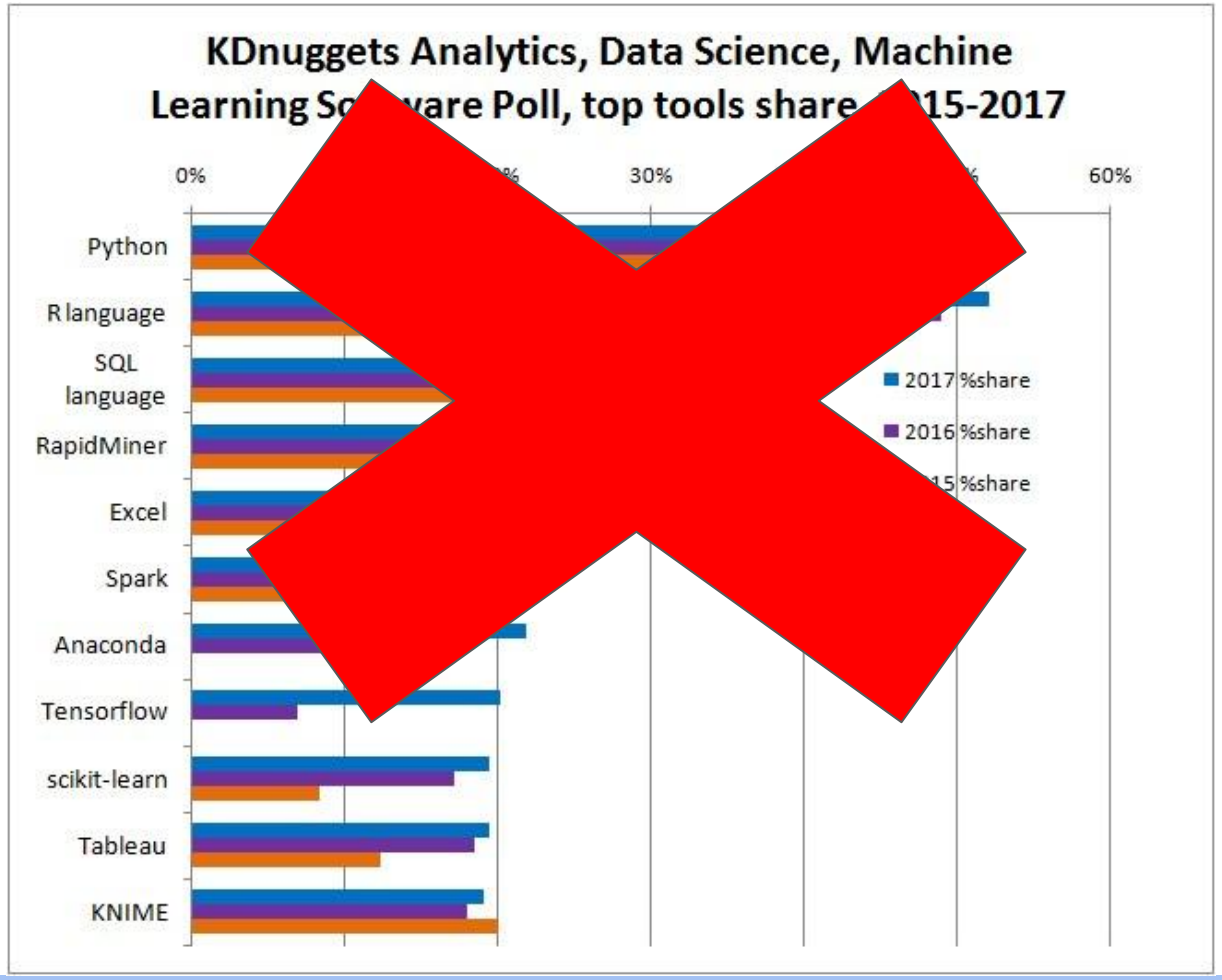
Сырые данные: 200 000 запросов в секунду

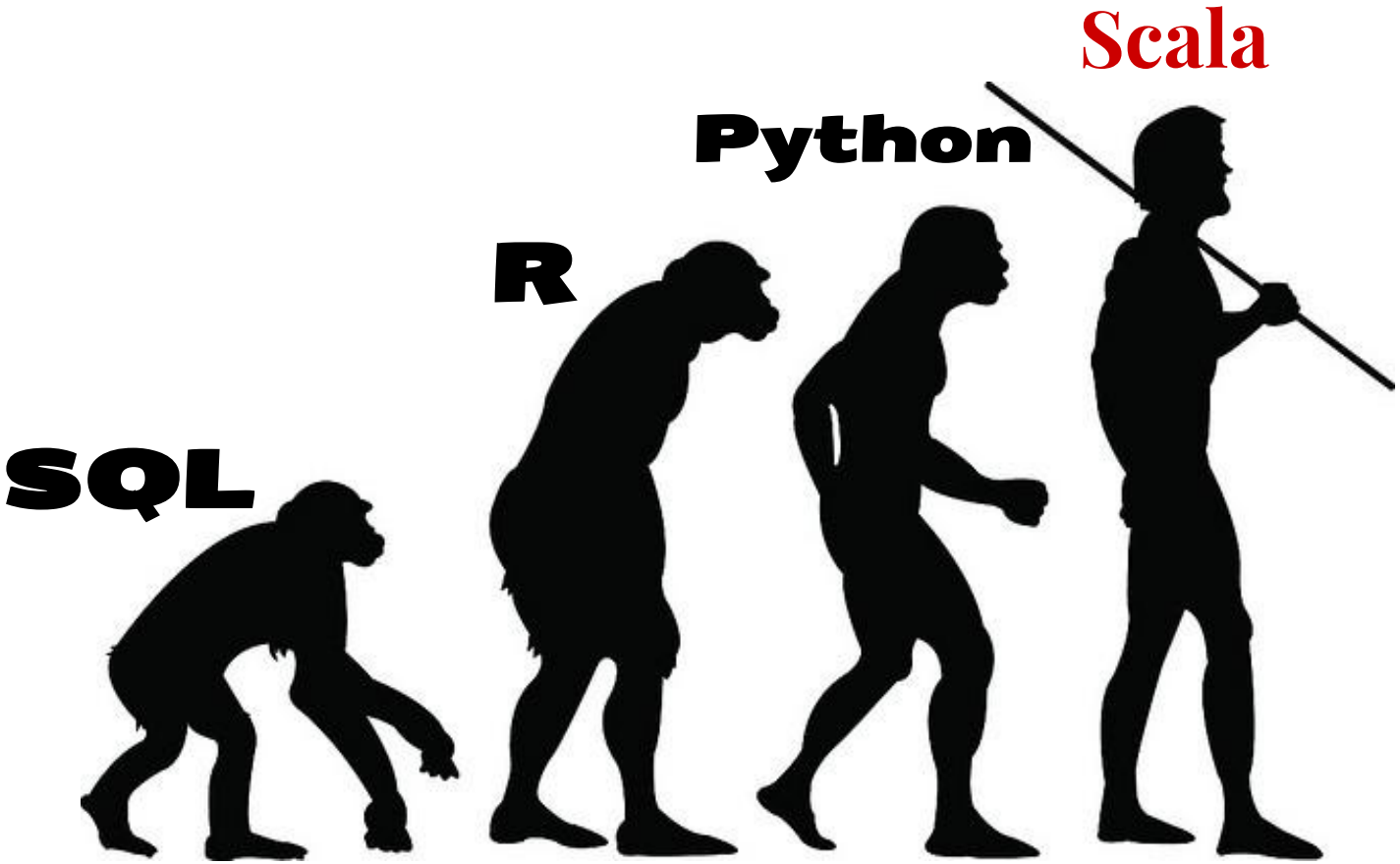
8 Терабайт в день



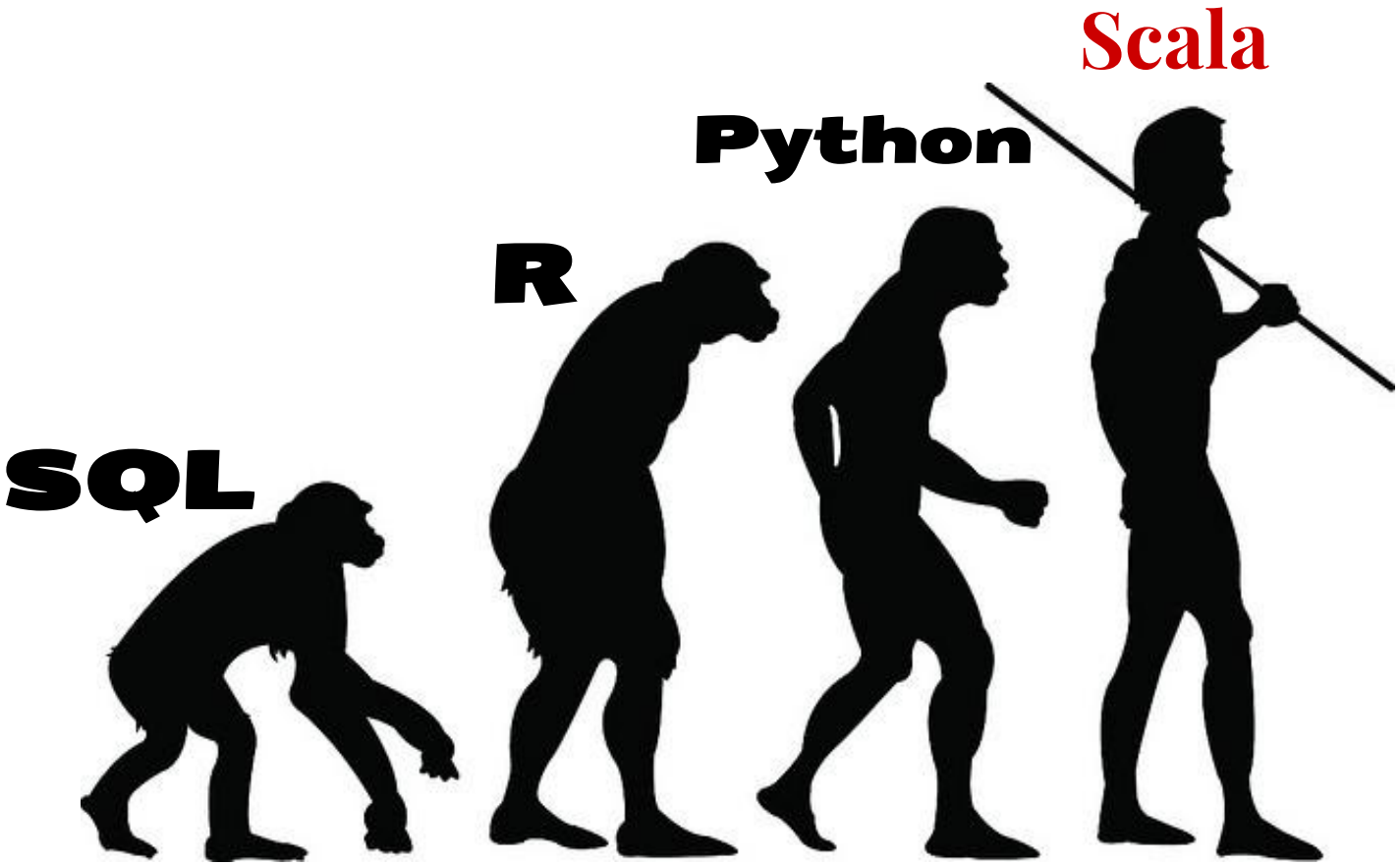
Сырые данные: 200 000 запросов в секунду

8 Терабайт в день

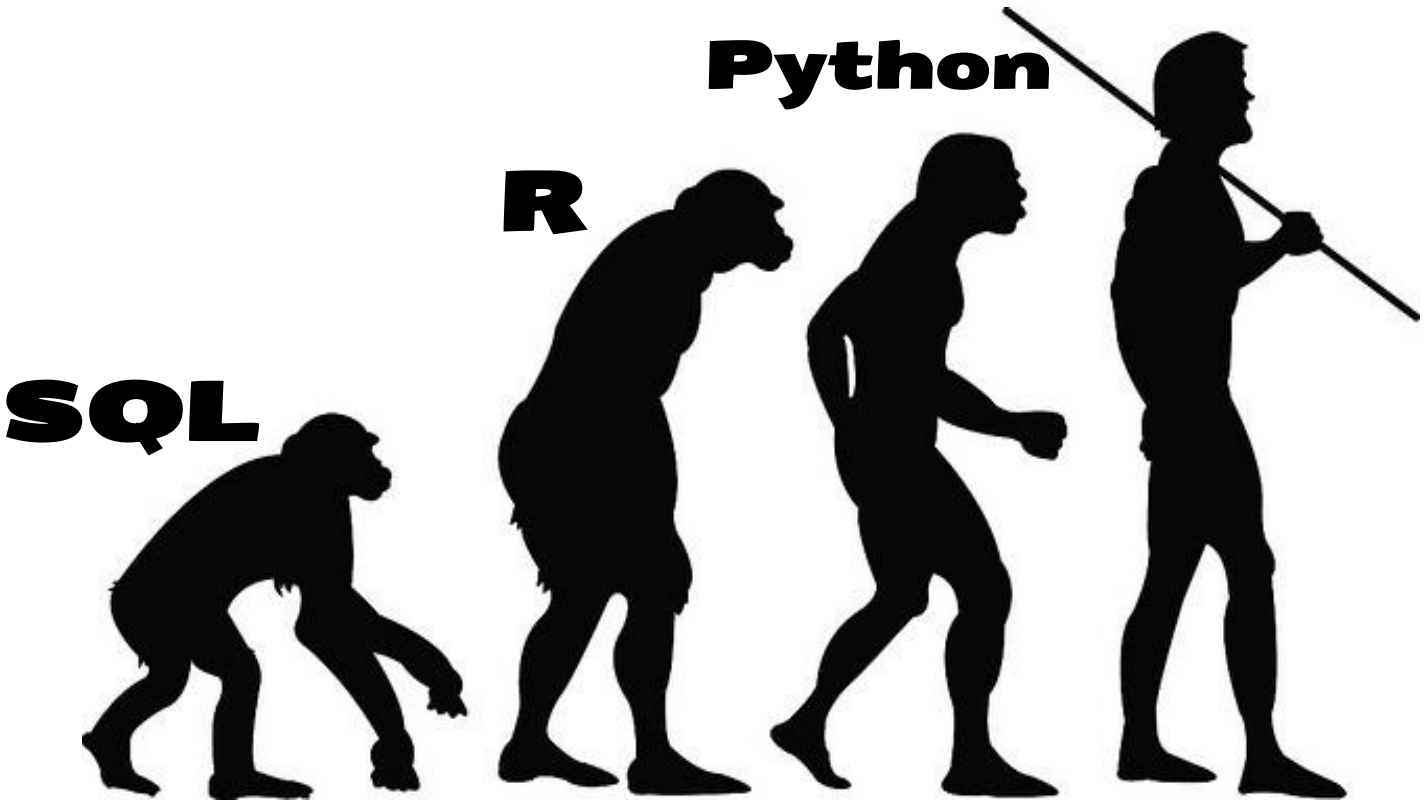



Scala
Python
R
SQL

Scala от SCALABLE



SCALABLE :
GROWABLE & ENABLING GROWTH



План .

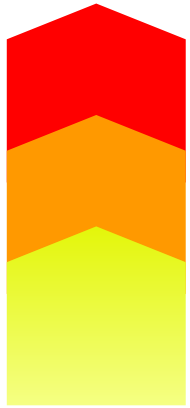
1. **Почему Scala ?**
2. **Обзор библиотек Data-Science на Scala**
3. **Что за проблему решаем сегодня?**
4. **Замарать руки в коде**
5. **Подвести итоги и выбрать**

подходящую нам библиотеку!

Ограничьте пространство поиска!

Какую библиотеку выбрать?

	Spark	Smile	Breeze	Saddle
Обучающий /оптимизирующий алгоритм				
Математический статистический анализ				
Конфигурация/ оптимизация алгоритма				
Предобработка данных				
Оценка моделей и качества анализа				
Визуализация				

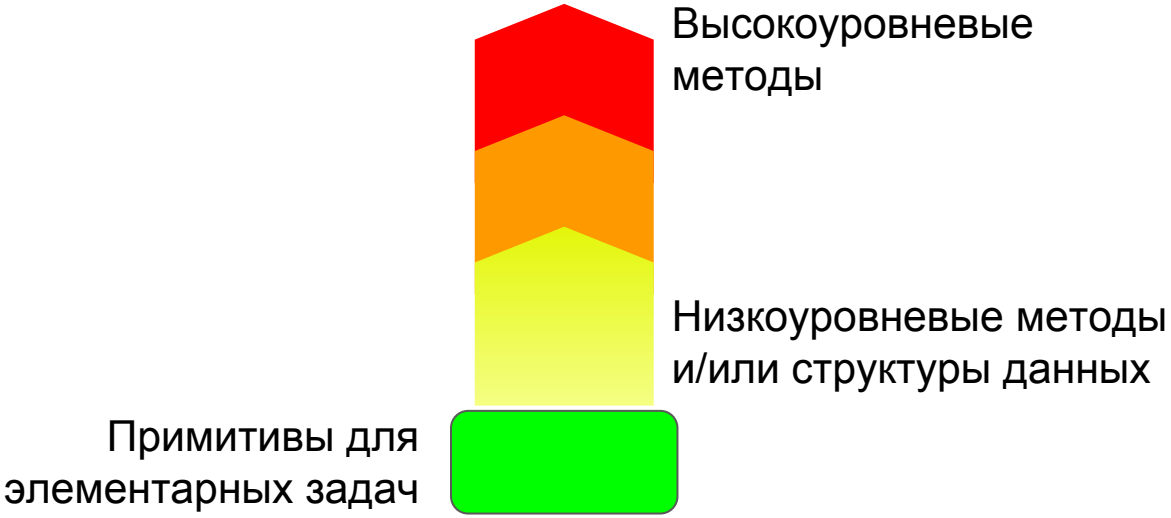


Примитивы для элементарных задач

Ограничьте пространство поиска!

Какую библиотеку выбрать?

	DeepLearning .scala (ThoughtWorks)	Neuron	DeepLearning4j	BigDL (on top of Spark)
Глубокое Обучение				

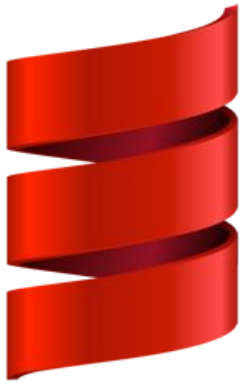


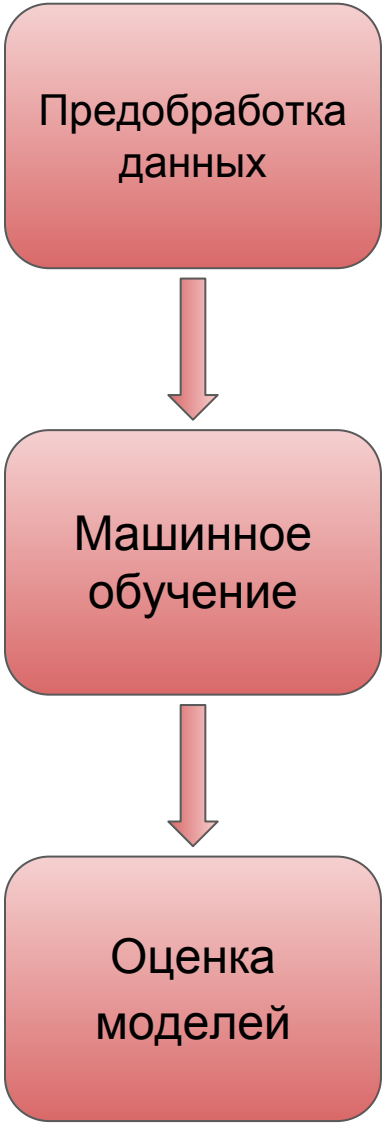
Scala

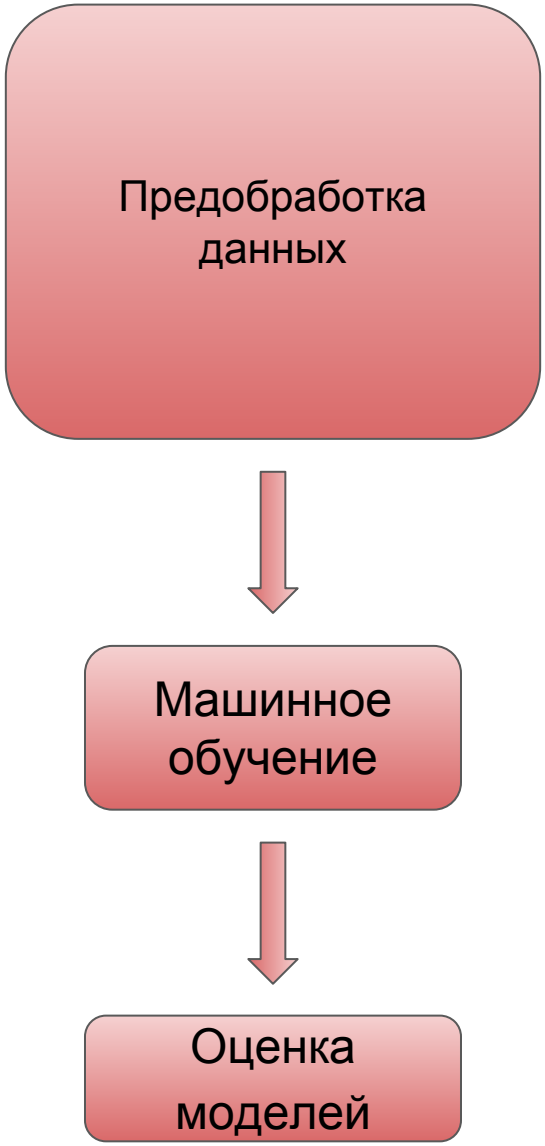
Spark MLlib/SQL

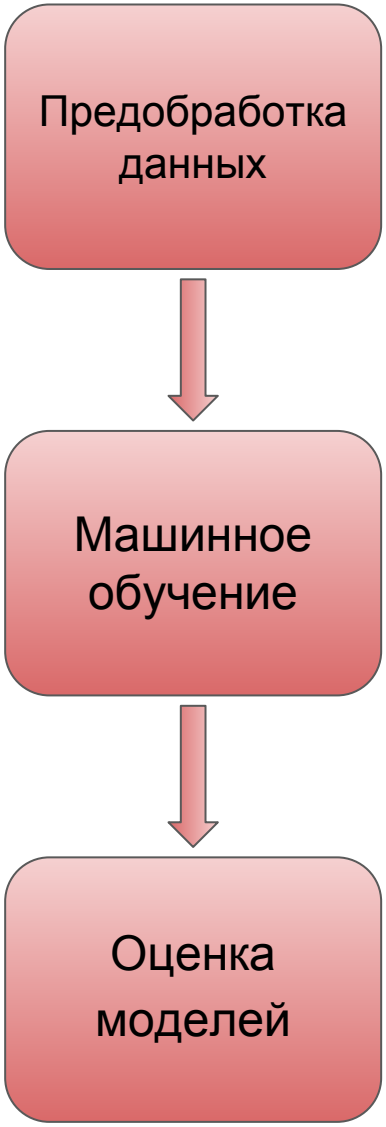
SMILE

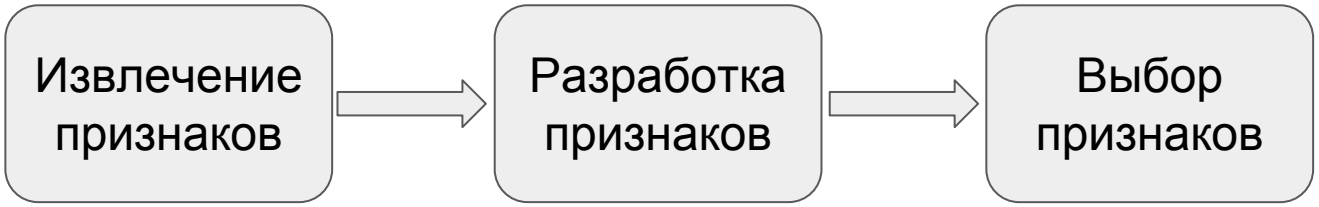
Saddle

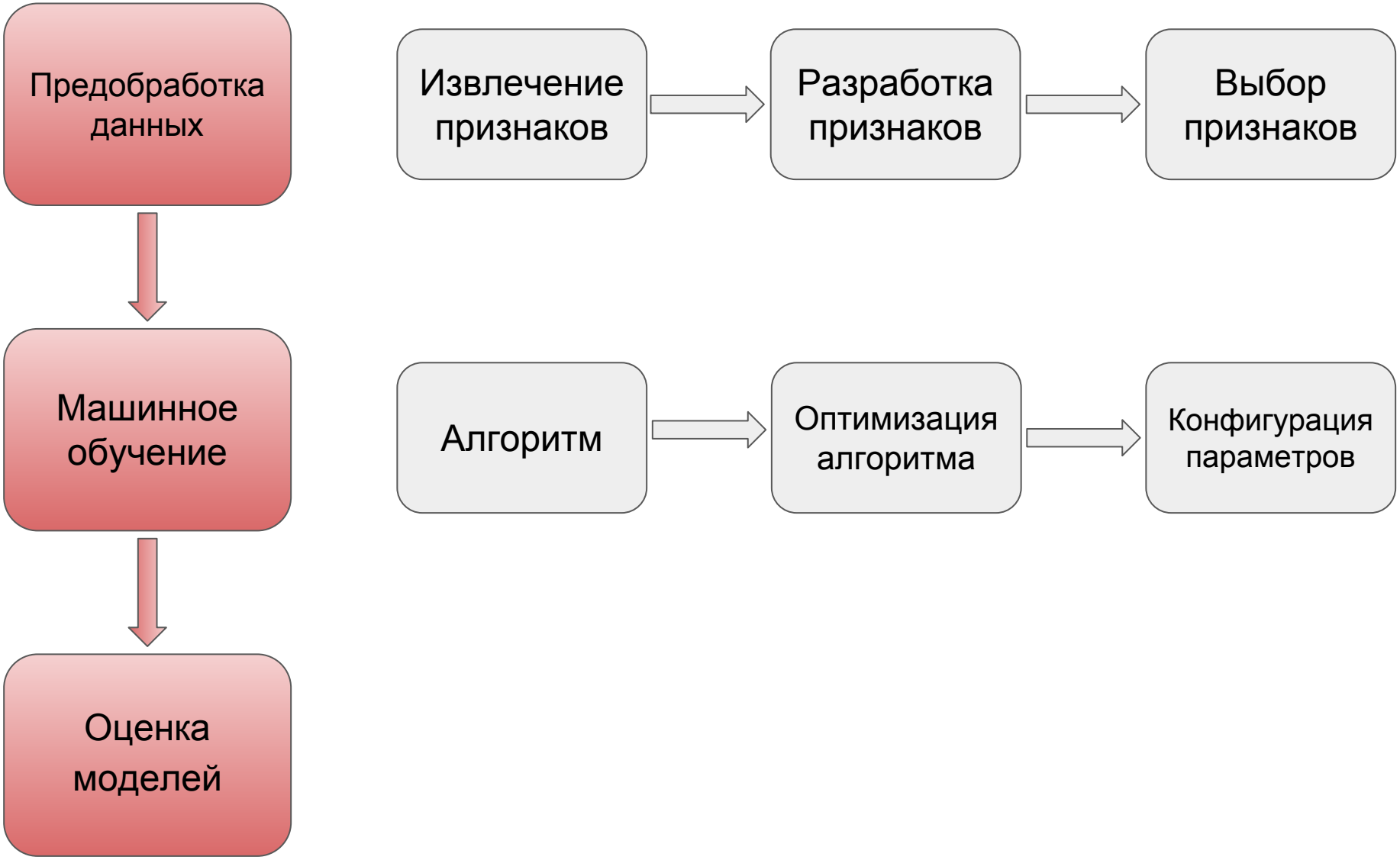


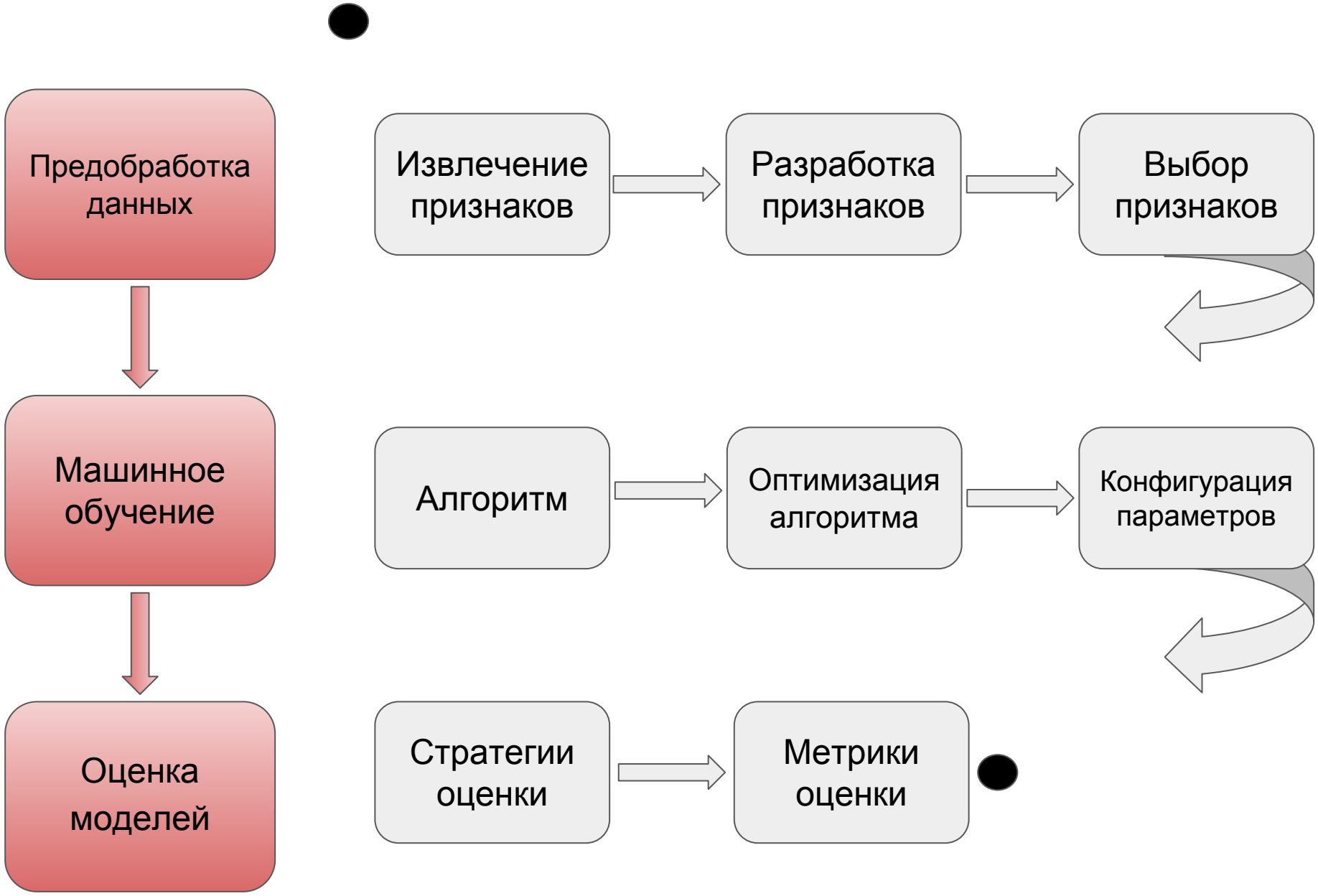

Предобработка данных
Машинное обучение
Оценка моделей

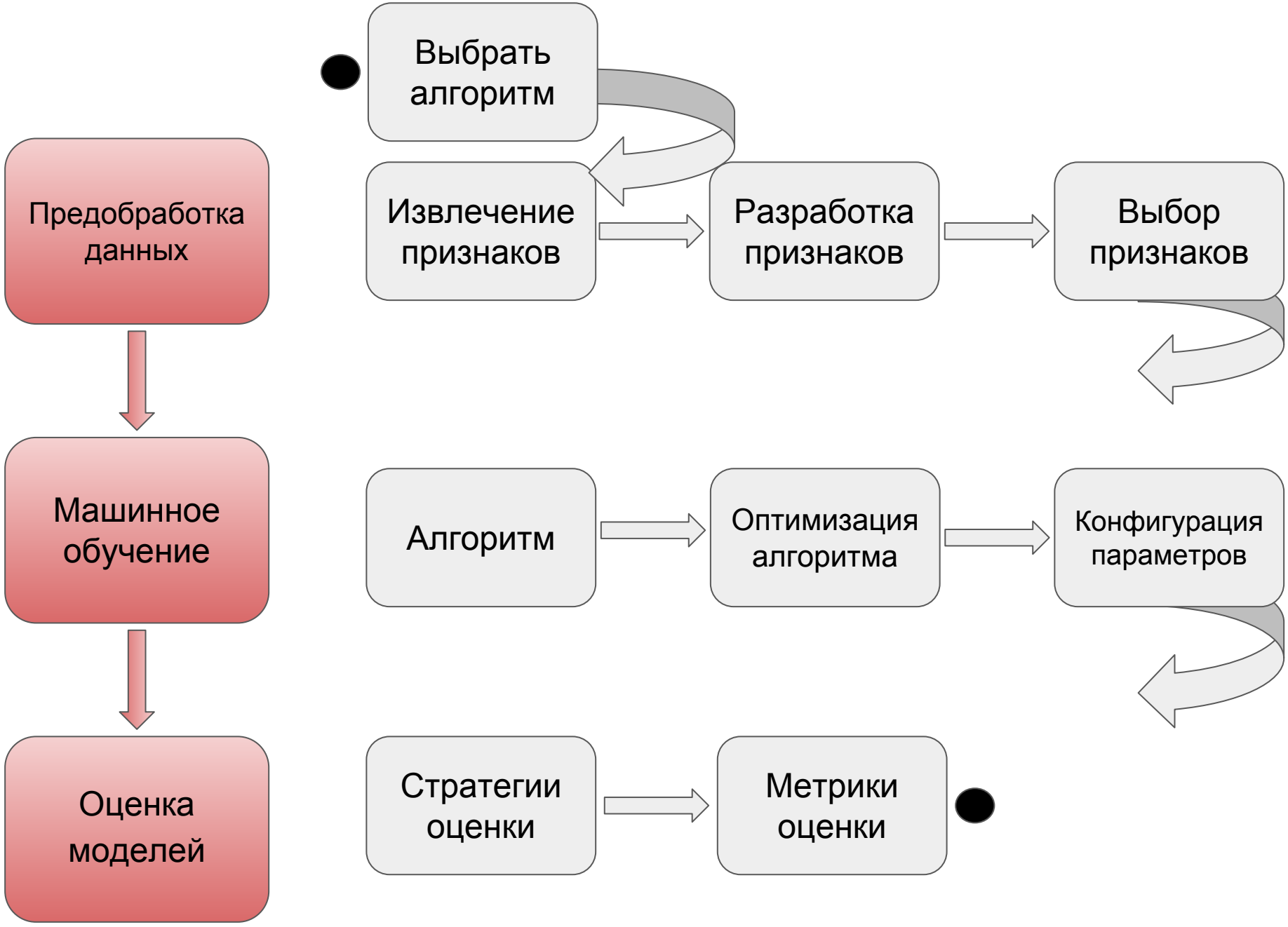

Предобработка данных
Машинное обучение
Оценка моделей


Предобработка данных
Машинное обучение
Оценка моделей

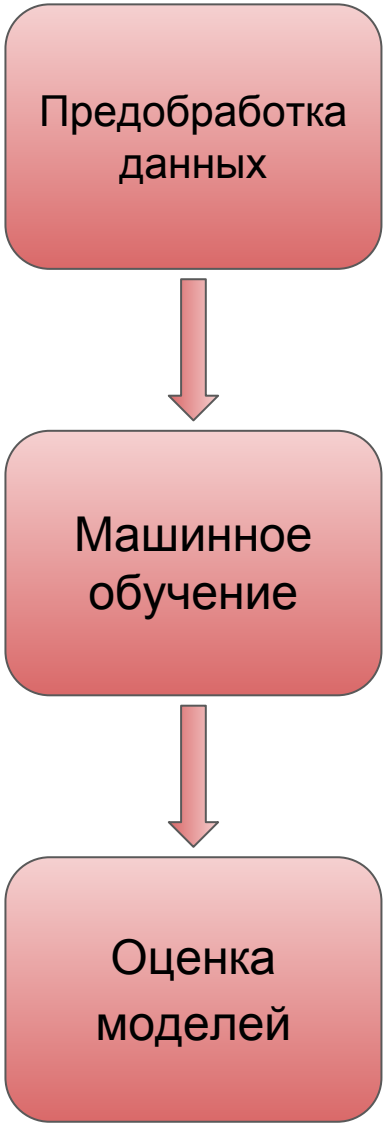

Извлечение признаков
Разработка признаков
Выбор признаков


Предобработка данных
Машинное обучение
Оценка моделей
Извлечение признаков
Разработка признаков
Выбор признаков
Алгоритм
Оптимизация алгоритма
Конфигурация параметров


Предобработка данных
Машинное обучение
Оценка моделей
Извлечение признаков
Разработка признаков
Выбор признаков
Алгоритм
Оптимизация алгоритма
Конфигурация параметров
Стратегии оценки
Метрики оценки


Выбрать алгоритм
Предобработка данных
Извлечение признаков
Разработка признаков
Выбор признаков
Машинное обучение
Алгоритм
Оптимизация алгоритма
Конфигурация параметров
Оценка моделей
Стратегии оценки
Метрики оценки

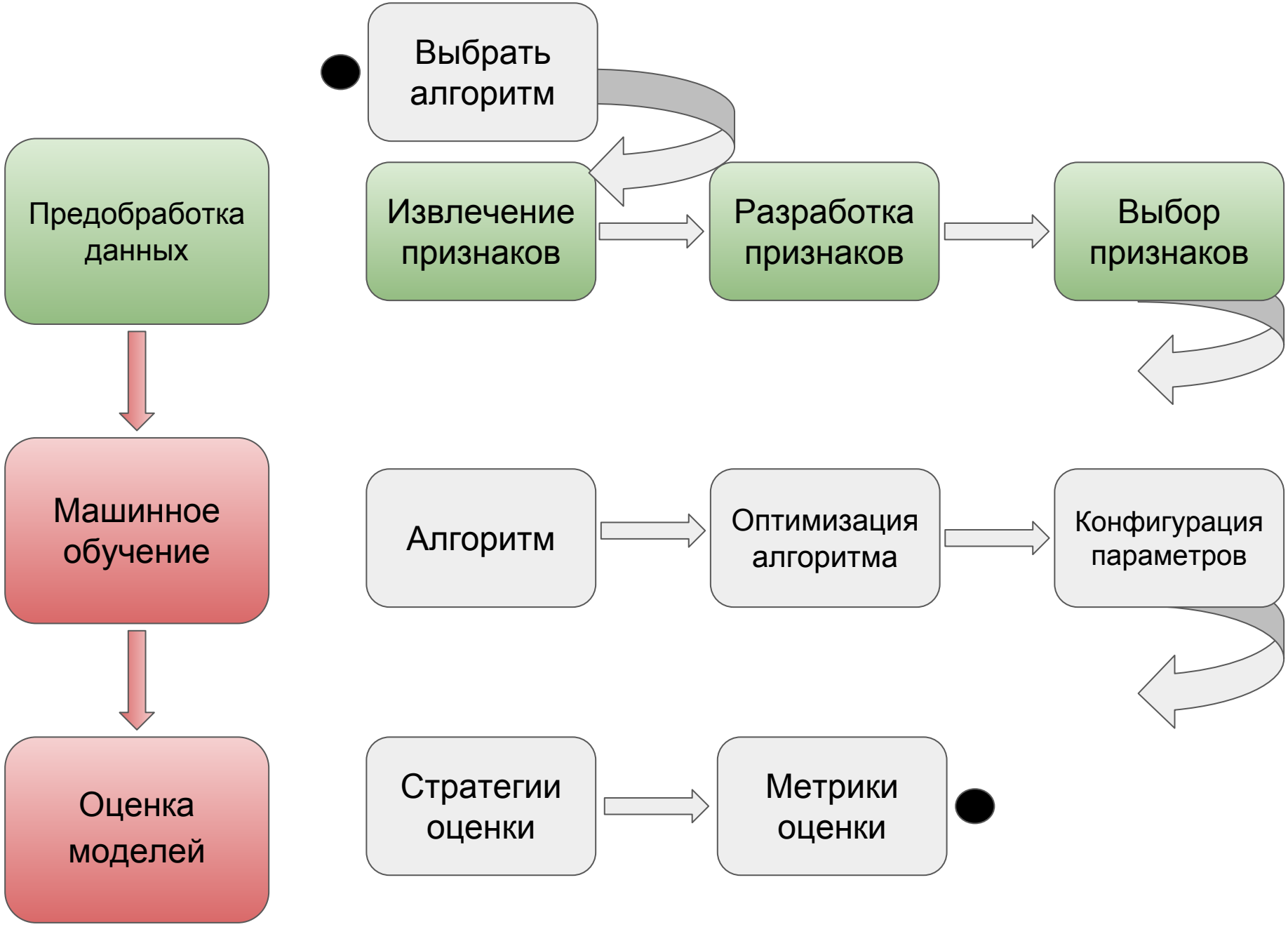
Spark MLlib/SQL



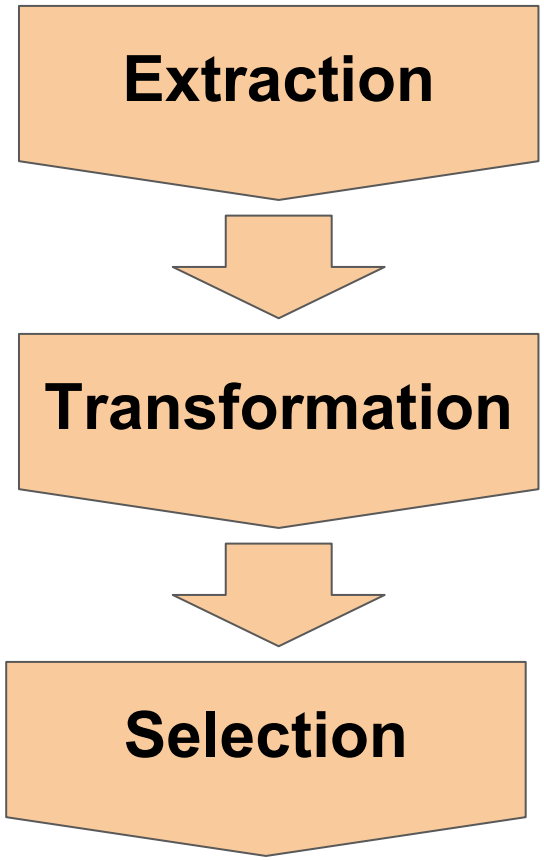
Saddle

Spark MLlib/SQL

SMILE


Предобработка данных
Машинное обучение
Оценка моделей
Выбрать алгоритм
Извлечение признаков
Разработка признаков
Выбор признаков
Алгоритм
Оптимизация алгоритма
Конфигурация параметров
Стратегии оценки
Метрики оценки

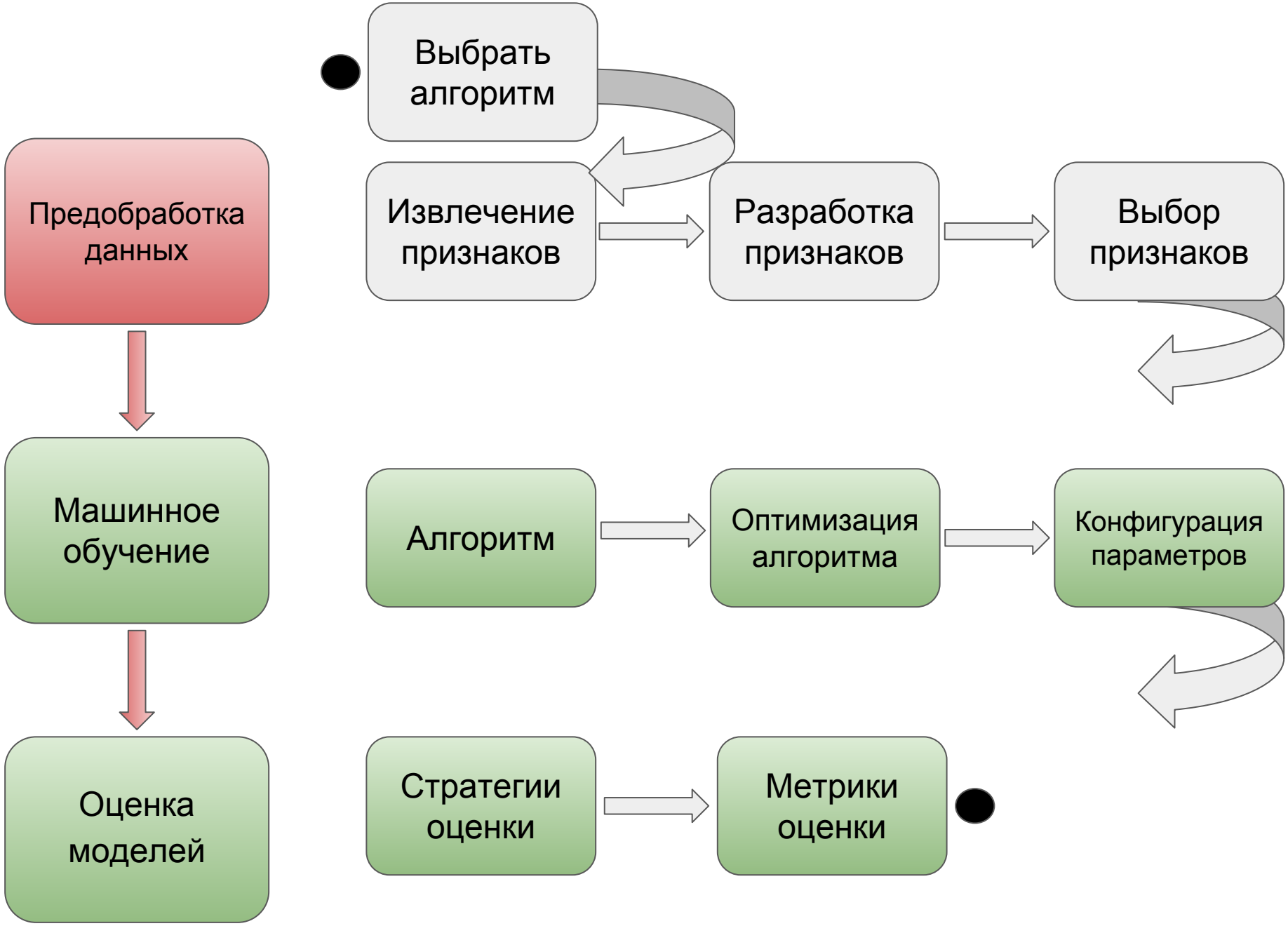
Предобработка данных: **Spark MLlib**



Предобработка данных: **Saddle**

- Векторы
- Матрицы
- Серии
- Фреймы
- Индексы

Class	Description
Vec	1D vector
Mat	2D matrix
Series	1D indexed vector
Frame	2D indexed matrix
Index	Hashmap-like

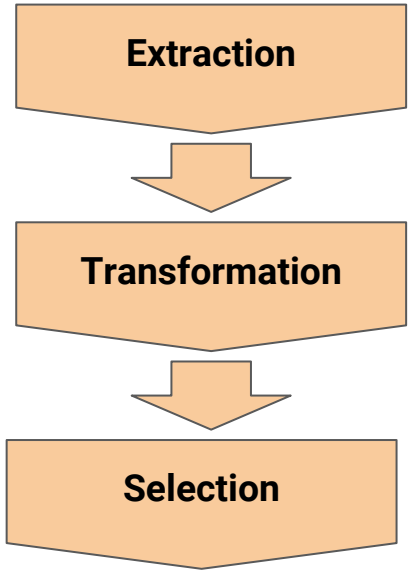

Выбрать алгоритм
Предобработка данных
Извлечение признаков
Разработка признаков
Выбор признаков
Машинное обучение
Алгоритм
Оптимизация алгоритма
Конфигурация параметров
Оценка моделей
Стратегии оценки
Метрики оценки

Обучение данных: **Spark MLlib**

- Классификация
- Кластеризация
- Регрессионный анализ
- Линейные методы
- Алгоритмы на основе деревьев
- Обработка естественного языка
- ……..и многие другие!

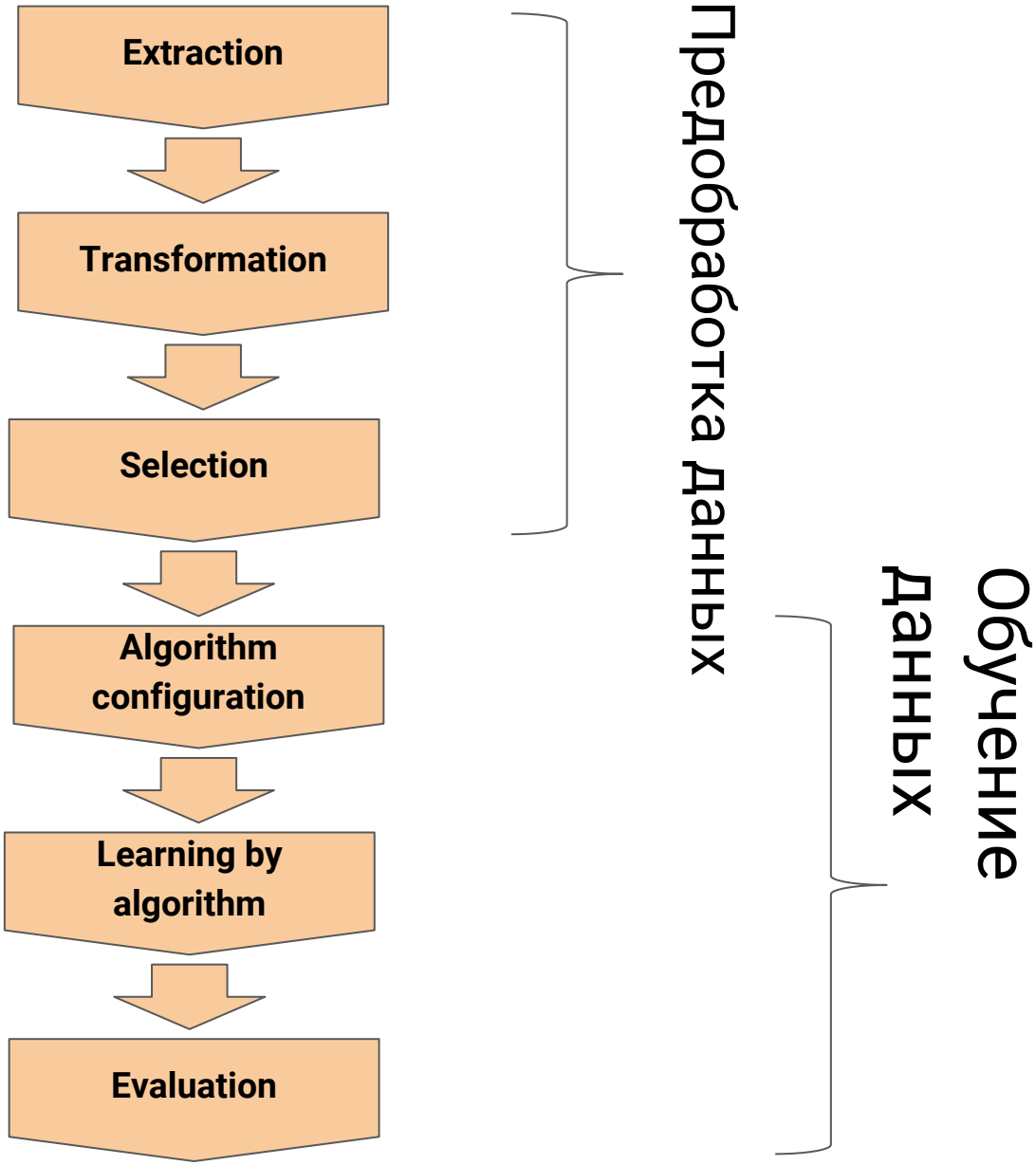
API на основе датафреймов

Spark MLlib : Pipelines



Предобработка данных

Spark MLlib : Pipelines

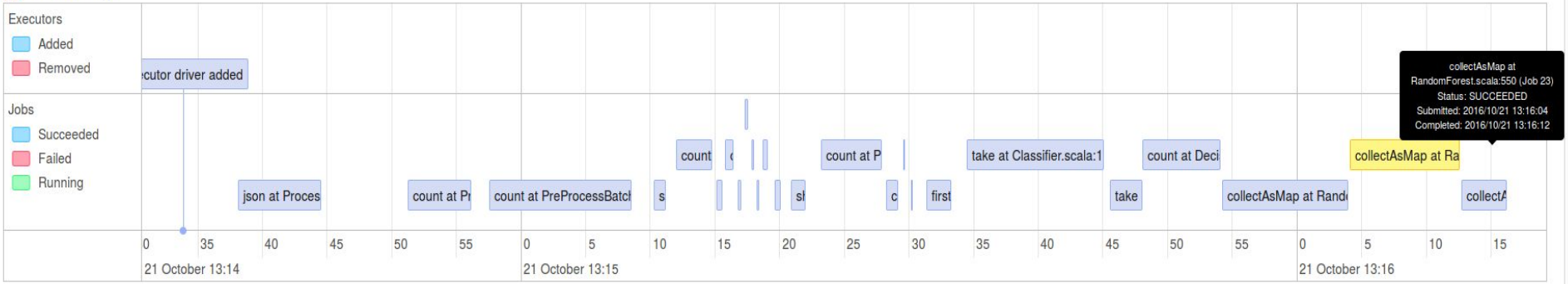


Spark MLlib : графический интерфейс

Spark Jobs (?)

User: anastasia
Total Uptime: 1.8 min
Scheduling Mode: FIFO
Active Jobs: 1
Completed Jobs: 24

▾ Event Timeline
Enable zooming



Active Jobs (1)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
24	collectAsMap at RandomForest.scala:550	2016/10/21 13:16:12	3 s	0/3	4/14

Completed Jobs (24)

Job Id	Description	Submitted	Duration	Stages: Succeeded/Total	Tasks (for all stages): Succeeded/Total
23	collectAsMap at RandomForest.scala:550	2016/10/21 13:16:04	9 s	2/2 (1 skipped)	10/10 (4 skipped)
22	collectAsMap at RandomForest.scala:894	2016/10/21 13:15:54	10 s	2/2 (1 skipped)	10/10 (4 skipped)
21	count at DecisionTreeMetadata.scala:116	2016/10/21 13:15:48	6 s	1/1 (1 skipped)	5/5 (4 skipped)
20	take at DecisionTreeMetadata.scala:112	2016/10/21 13:15:45	3 s	2/2	5/5
19	take at Classifier.scala:111	2016/10/21 13:15:34	11 s	3/3	10/10
18	first at VectorAssembler.scala:56	2016/10/21 13:15:31	2 s	2/2	5/5

Обучение данных: **SMILE**

- Классификация
- Кластеринг
- Регрессионный анализ
- Линейные методы
- Алгоритмы на основе деревьев
- Обработка естественного языка
- ……..и многие другие!

API на основе массивов

Обучение данных: **SMILE**

- ★ Vector Quantization
- ★ Association Rule Mining
- ★ Manifold learning
- ★ Multi-dimensional scaling

БОЛЬШЕ

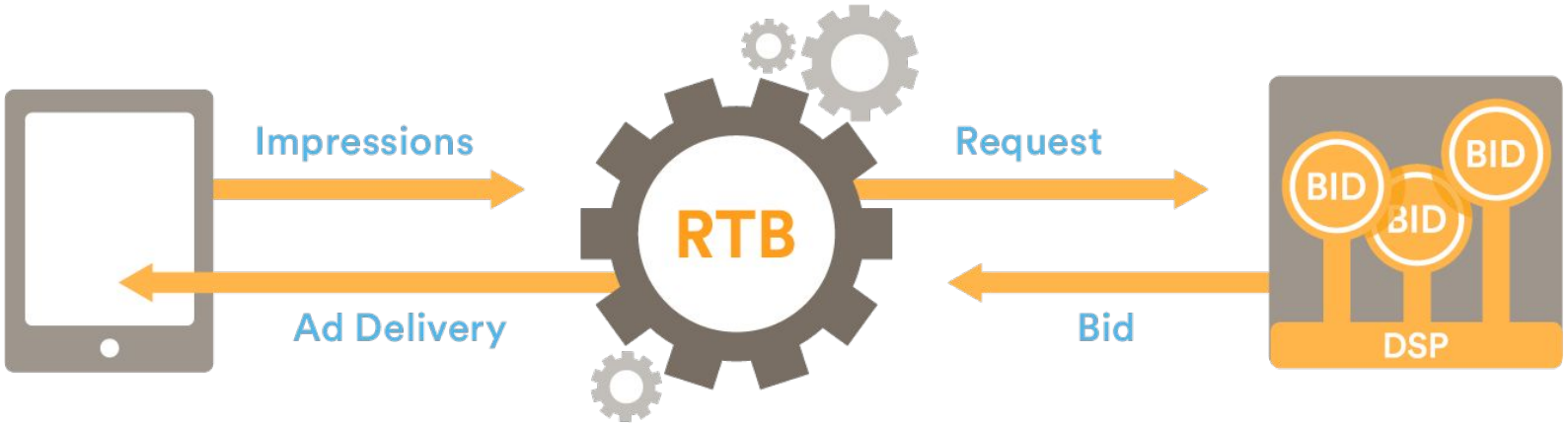
- ★ алгоритмов кластеризации
- ★ методов замещения отсутствующих данных

План .

1. **Почему Scala ?**
2. **Обзор библиотек Data-Science на Scala**
3. **Что за проблему решаем сегодня?**
4. **Замарать руки в коде**
5. **Подвести итоги и выбрать**

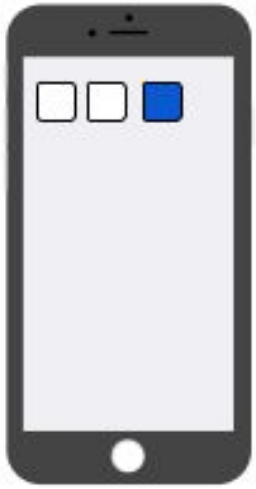
подходящую нам библиотеку!

Контекст : Технологии RTB



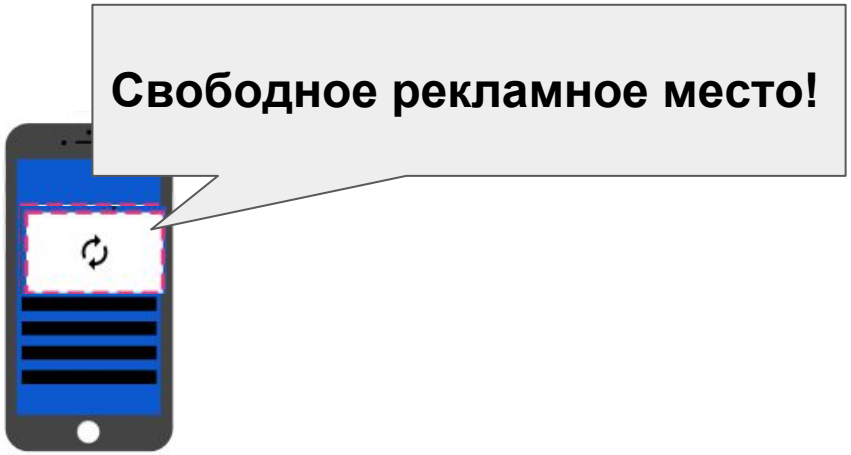
Технологии RTB

Вы открываете
мобильное приложение

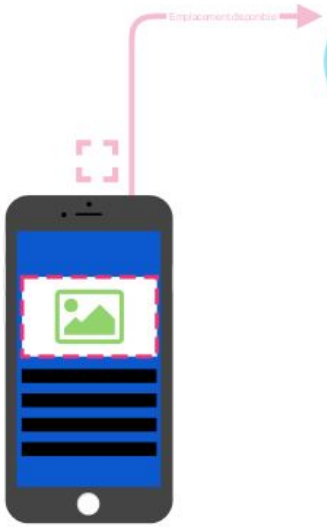


Технологии RTB

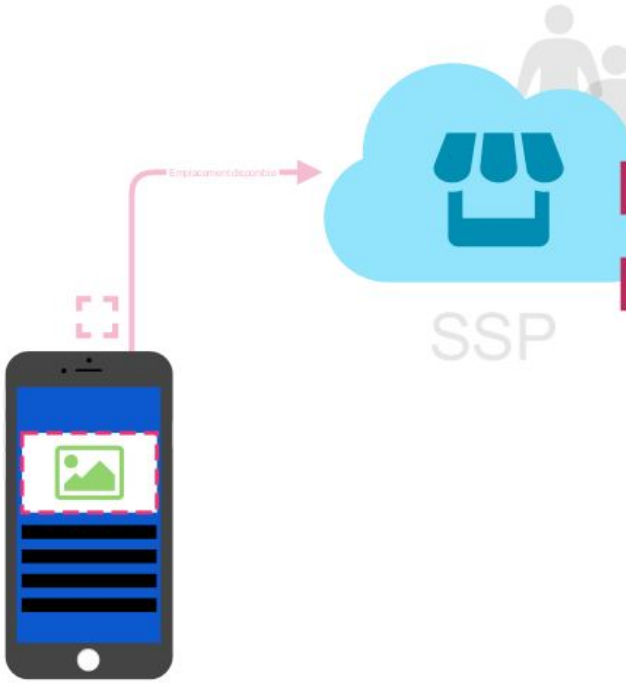
Технологии RTB



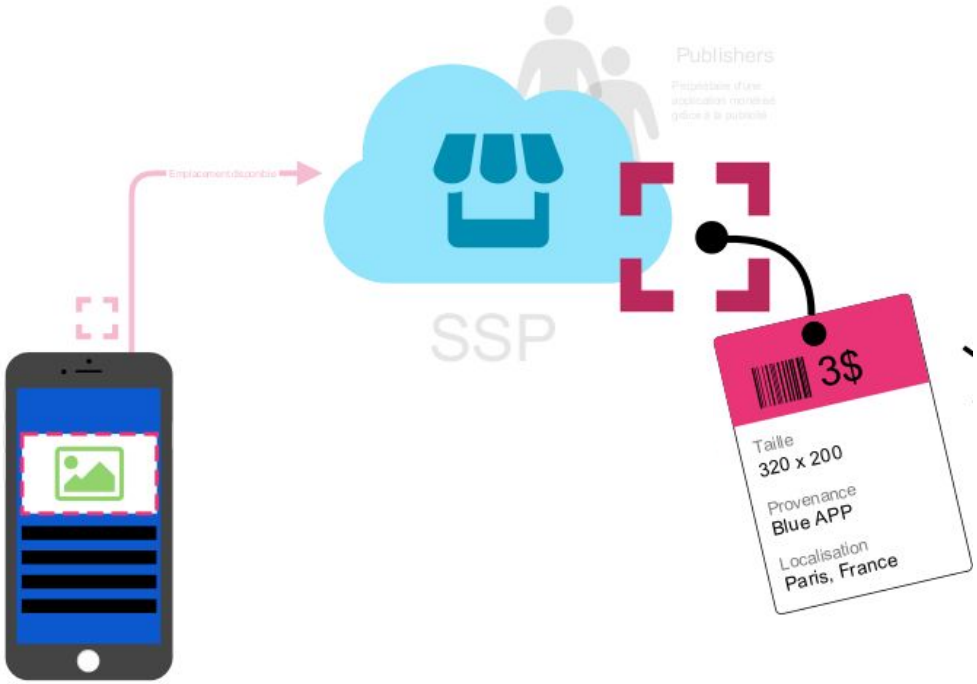
Технологии RTB



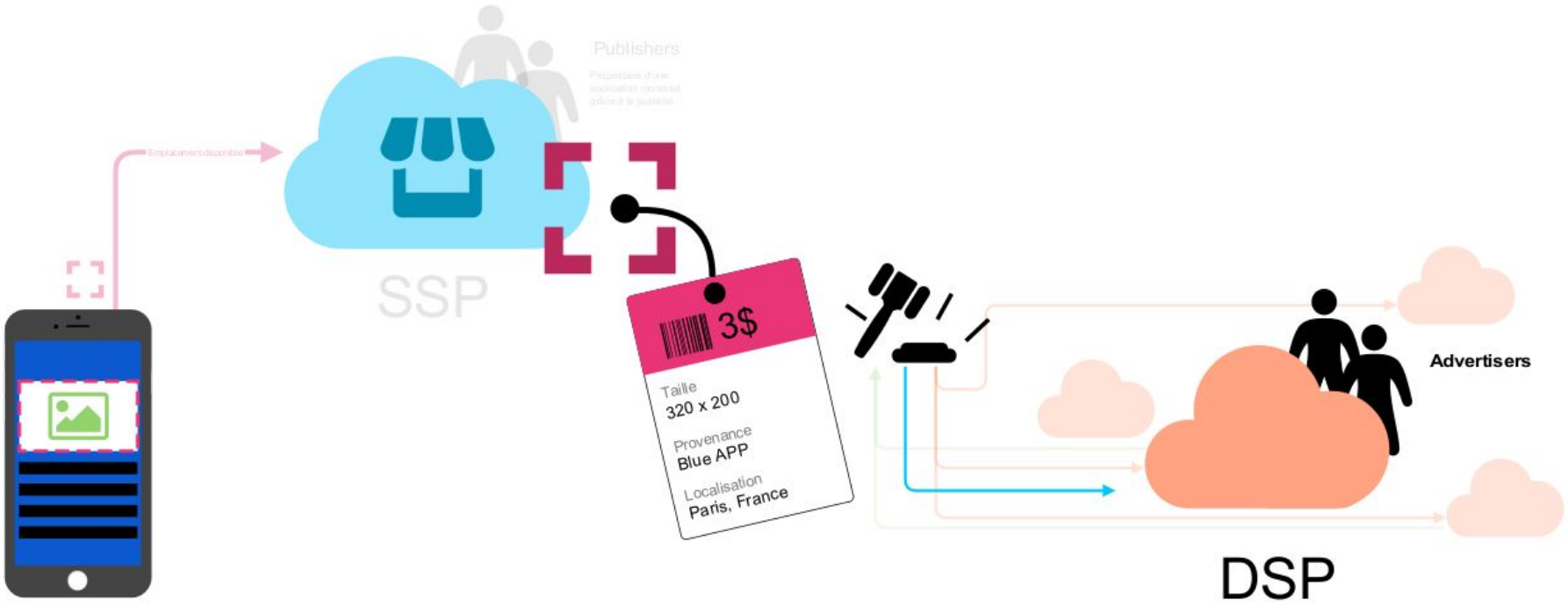
Технологии RTB

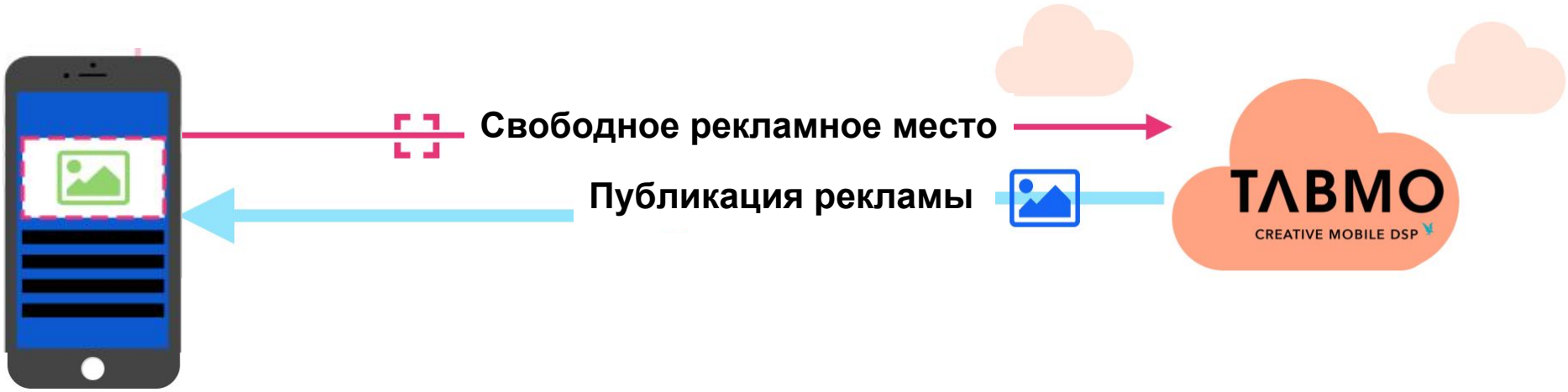


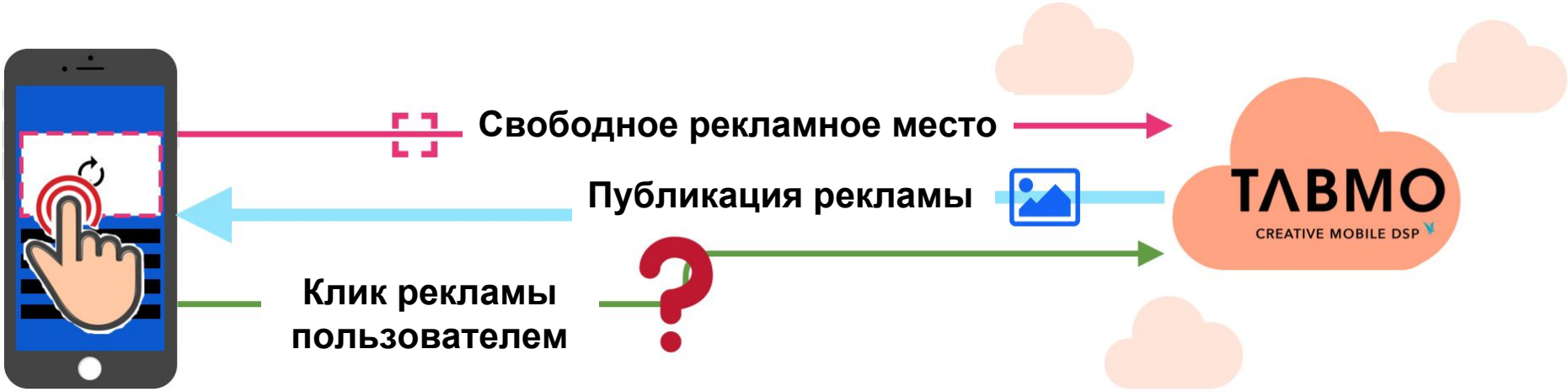
Технологии RTB



Технологии RTB

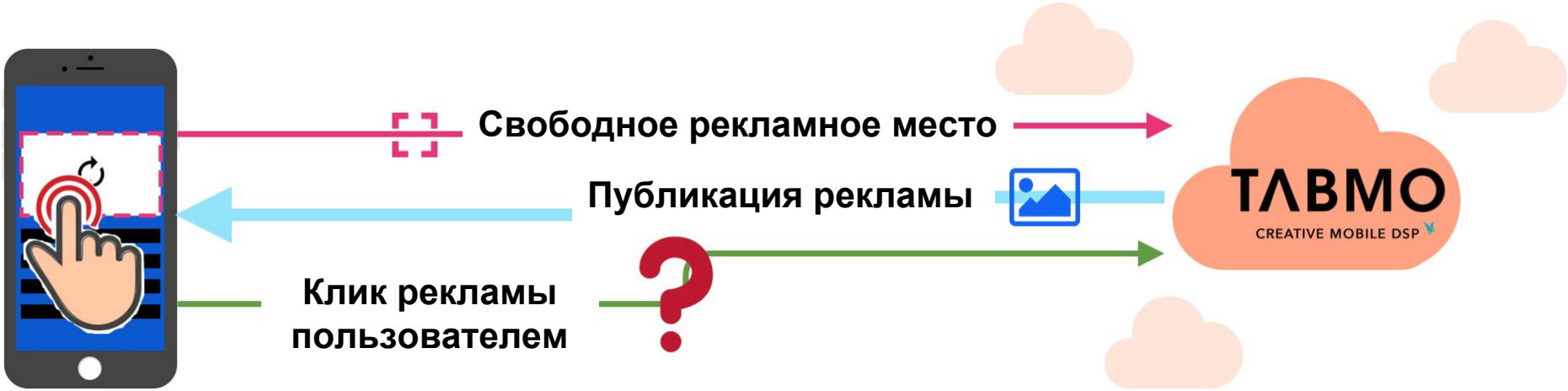



Свободное рекламное место
Публикация рекламы
TABMO
CREATIVE MOBILE DSP


Свободное рекламное место
Публикация рекламы
Клик рекламы пользователем
TABMO
CREATIVE MOBILE DSP

Задача:

Оптимизировать уровень кликов показанных реклам



Задача:

Оптимизировать уровень кликов афишируемых реклам

Нам нужно знать вероятность того,

что вы кликните на рекламу

на основании :

Задача:

Оптимизировать уровень кликов показанных реклам

Нам нужно знать вероятность того,

что вы кликните на рекламу

на основании:

- конфигурации запроса

Задача:

Оптимизировать уровень кликов показанных реклам

Нам нужно знать вероятность того,

что вы кликните на рекламу

на основании :

- конфигурации запроса
- креативного формата рекламы

Задача:

Оптимизировать уровень кликов показанных реклам

Нам нужно знать вероятность того,

что вы кликните на рекламу

на основании :

- конфигурации запроса
- креативного формата рекламы
- истории пользователя

Задача:

Оптимизировать уровень кликов показанных реклам

Нам нужно знать вероятность того,

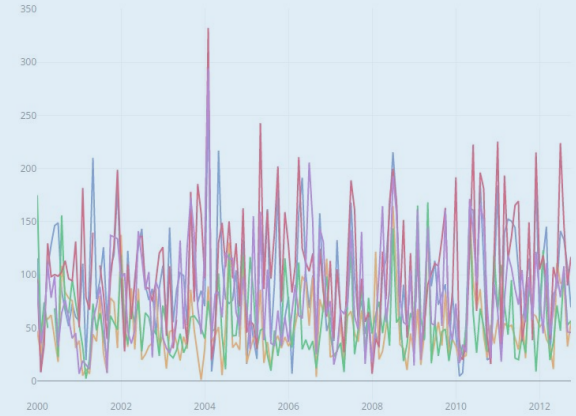
что вы кликните на рекламу

на основании:

- конфигурации запроса
- креативного формата рекламы
- истории пользователя
- сторонней и мета-информации

Определите задачу в контексте науки о данных !

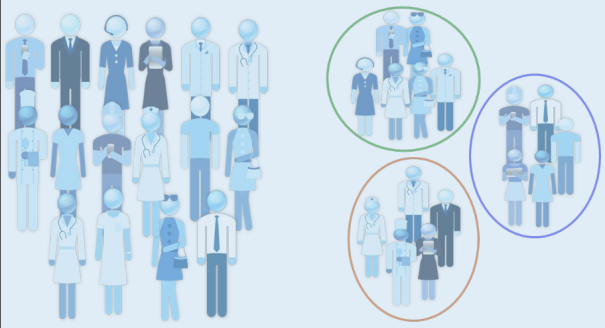
Анализ временных рядов



Классификация

Регрессионный

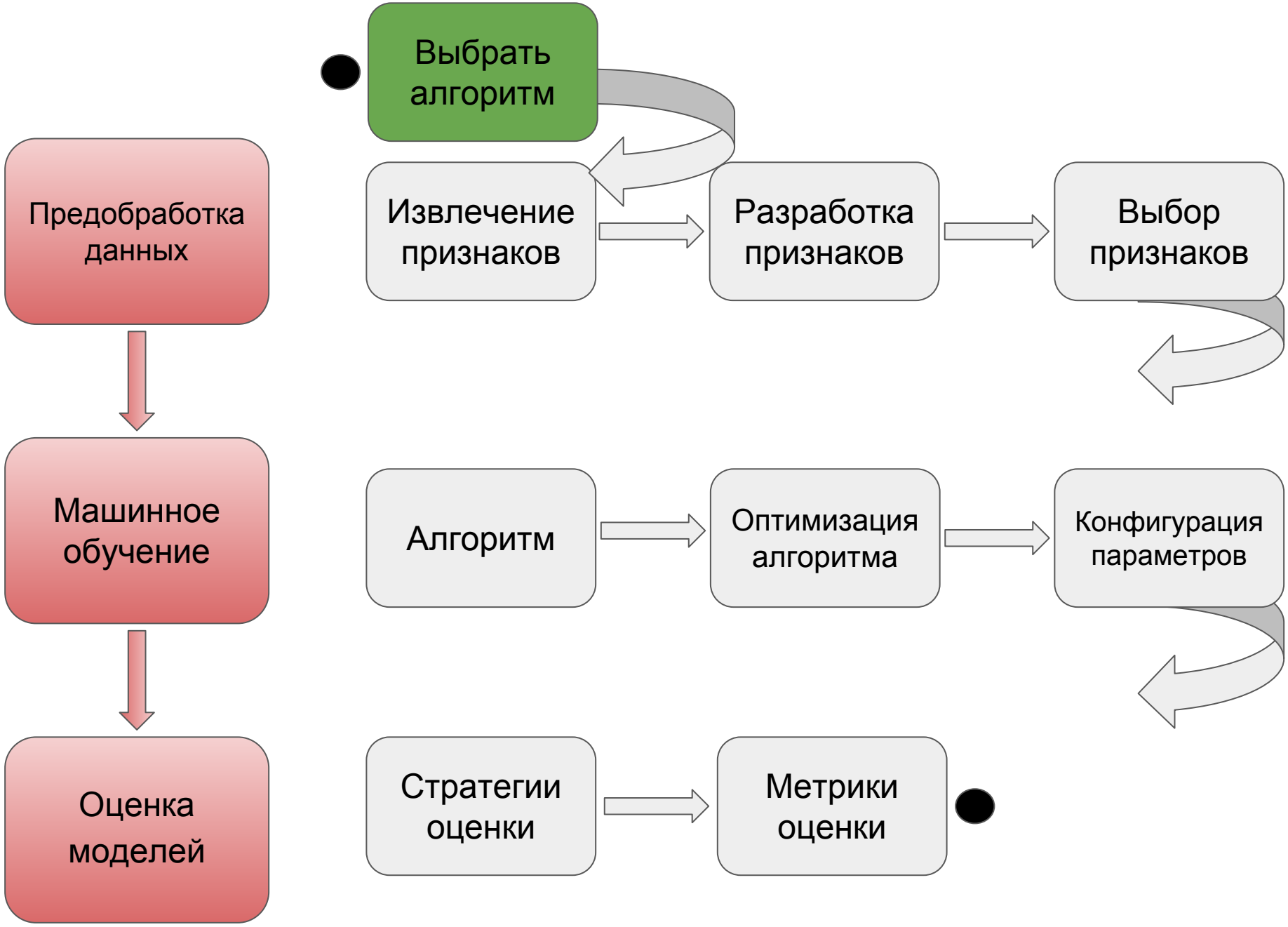
Кластеризация



~~собака~~
клик

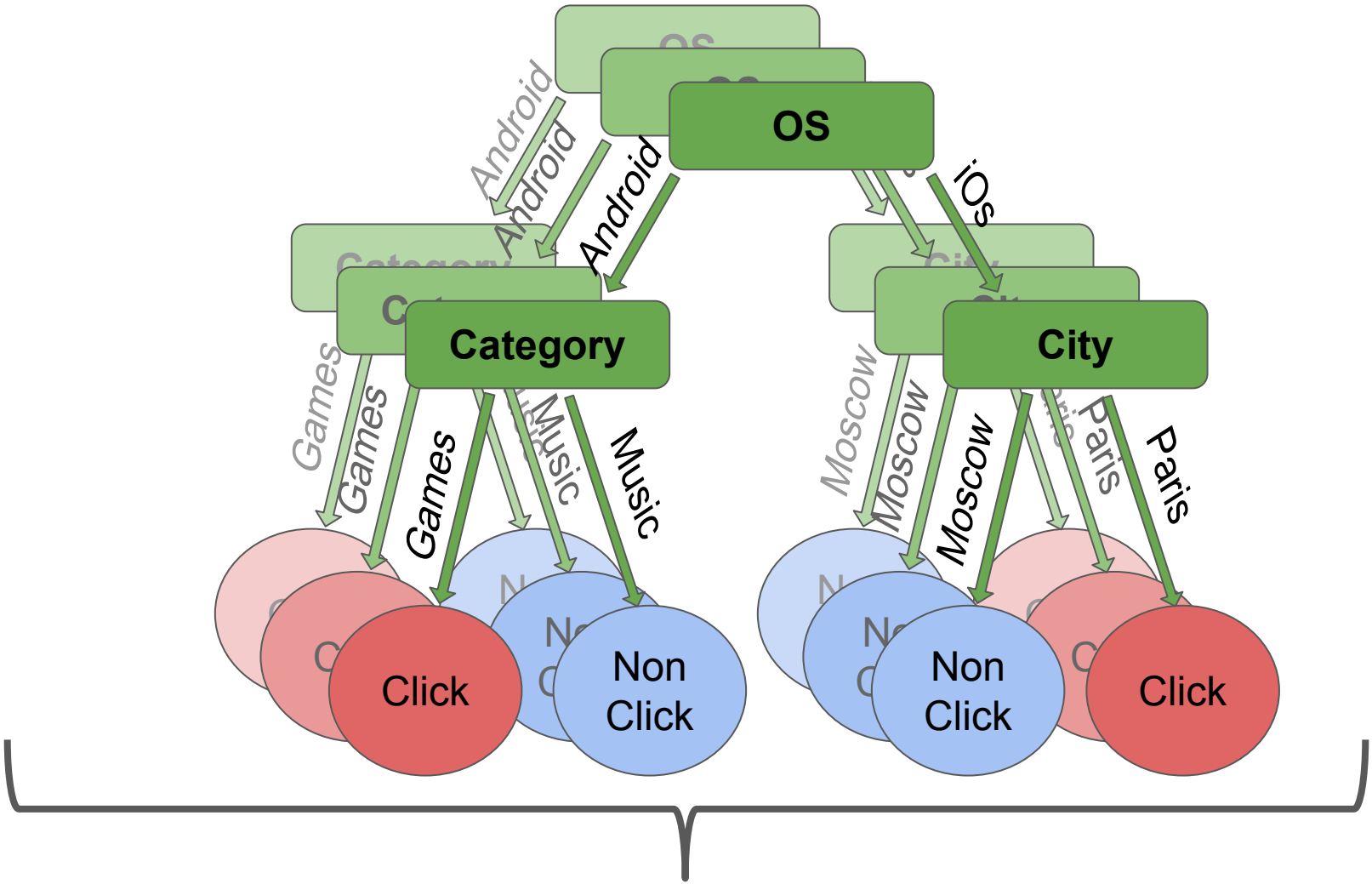
тация
аномал

~~кошка~~
нет клика


Выбрать алгоритм
Предобработка данных
Извлечение признаков
Разработка признаков
Выбор признаков
Машинное обучение
Алгоритм
Оптимизация алгоритма
Конфигурация параметров
Оценка моделей
Стратегии оценки
Метрики оценки

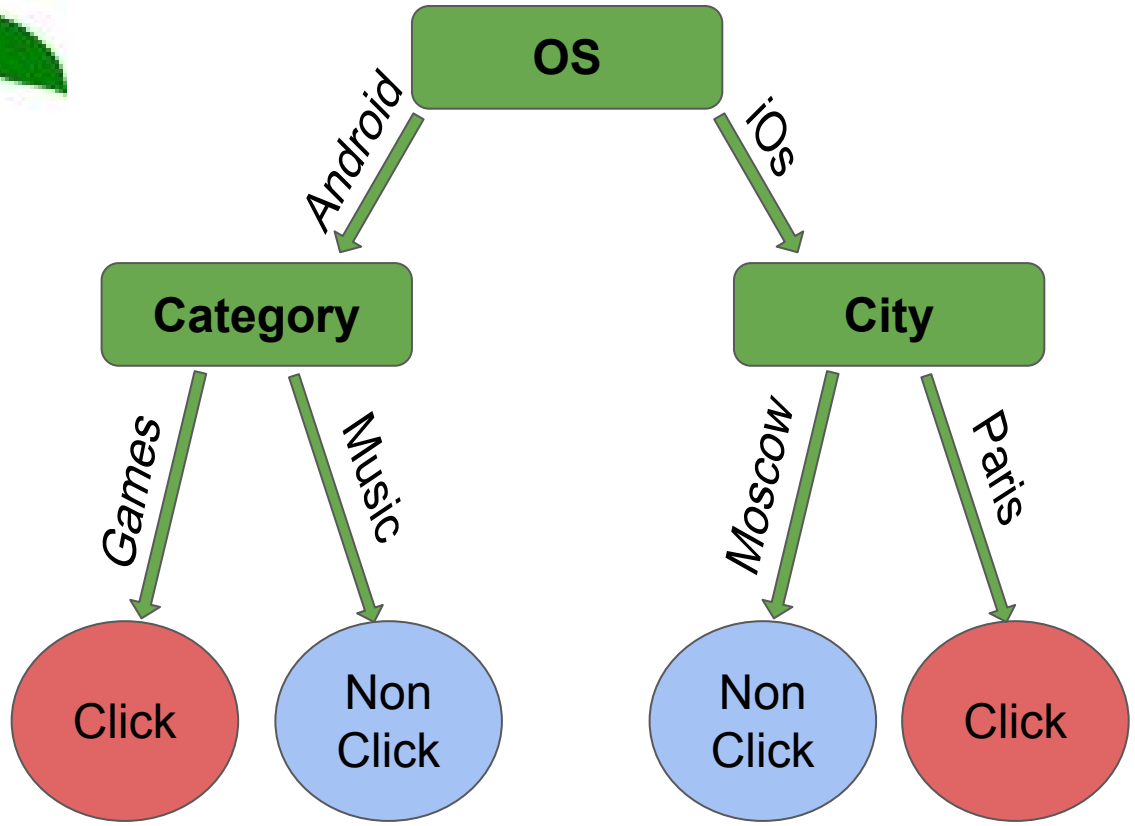
Случайный лес (Random Forest)

Алгоритм: **случайный лес** *(Random Forest)*



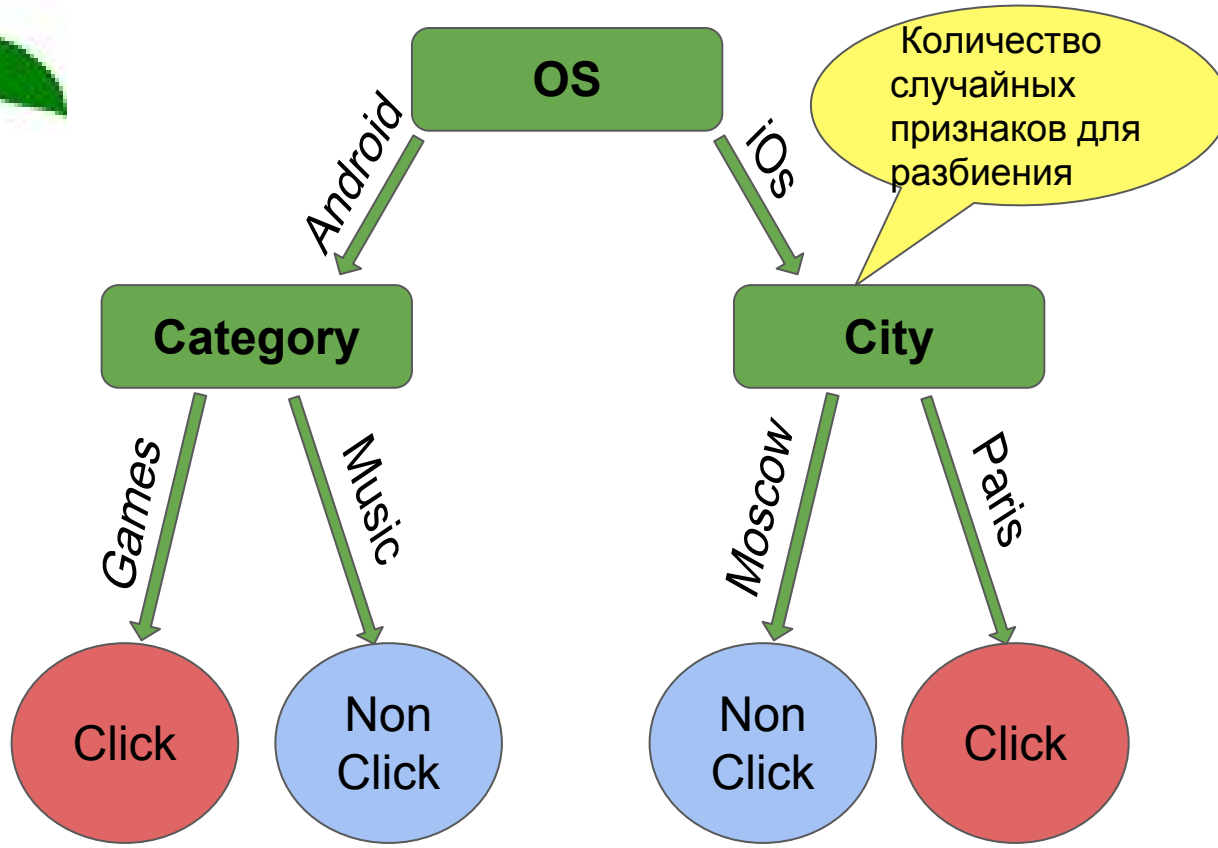
Алгоритм: **случайный лес** *(Random Forest)*

Дерево решений



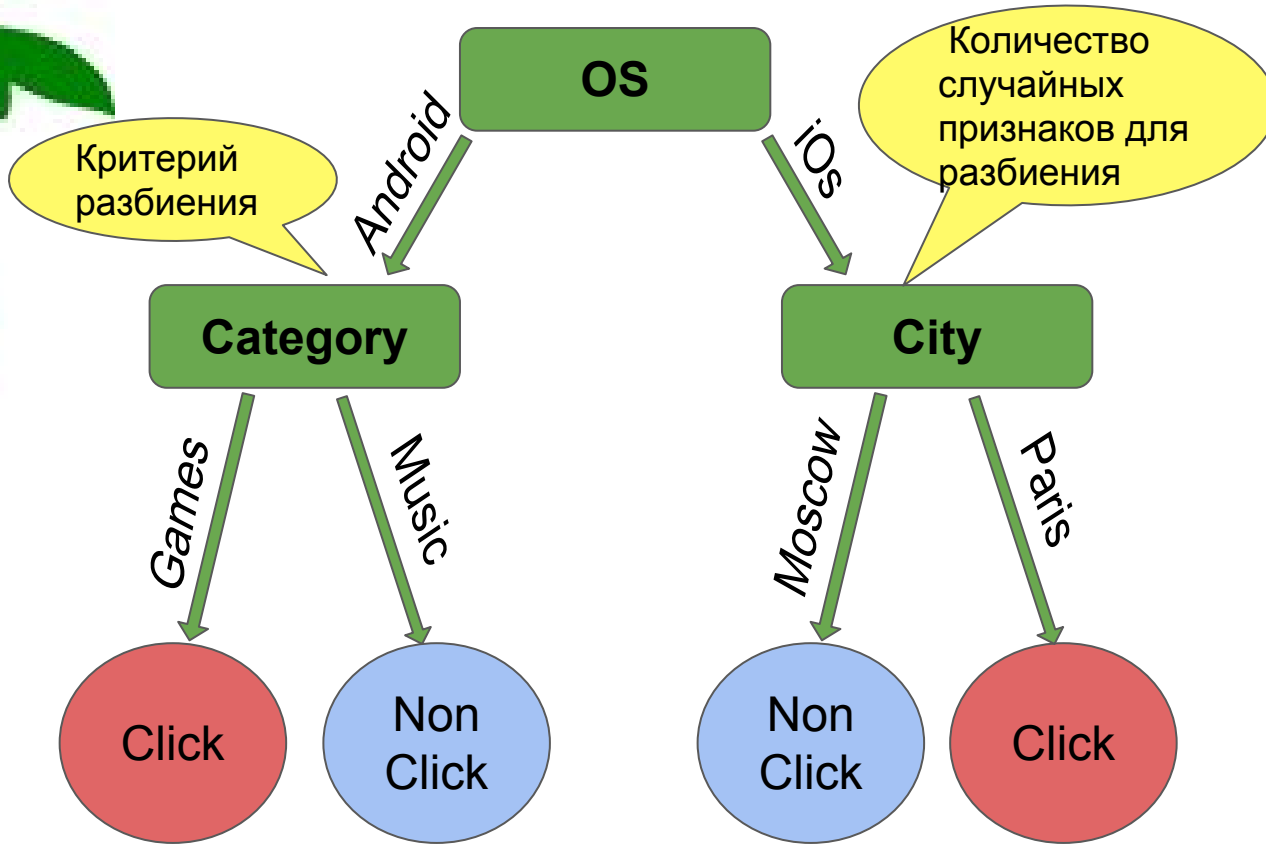
Алгоритм: **случайный лес** *(Random Forest)*

Дерево решений



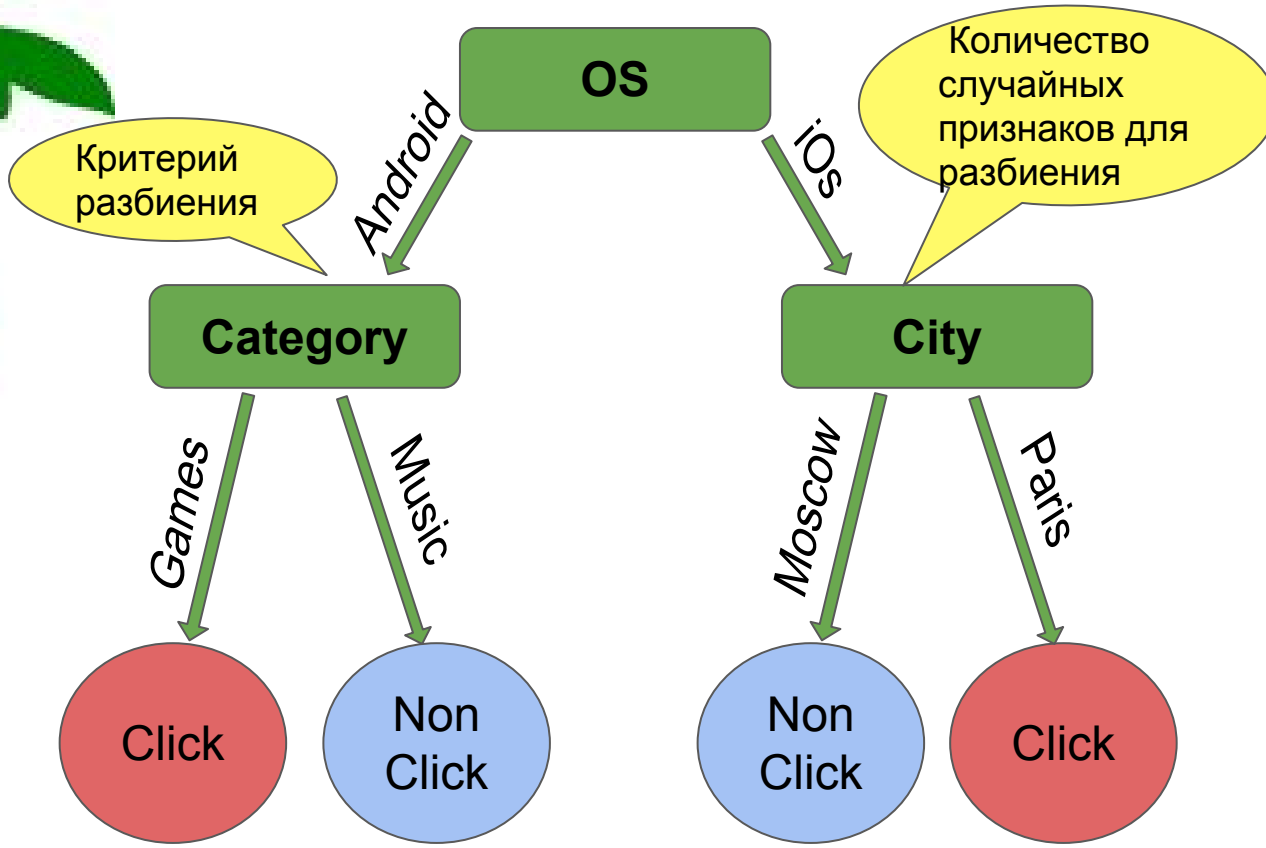
Алгоритм: **случайный лес** *(Random Forest)*

Дерево решений

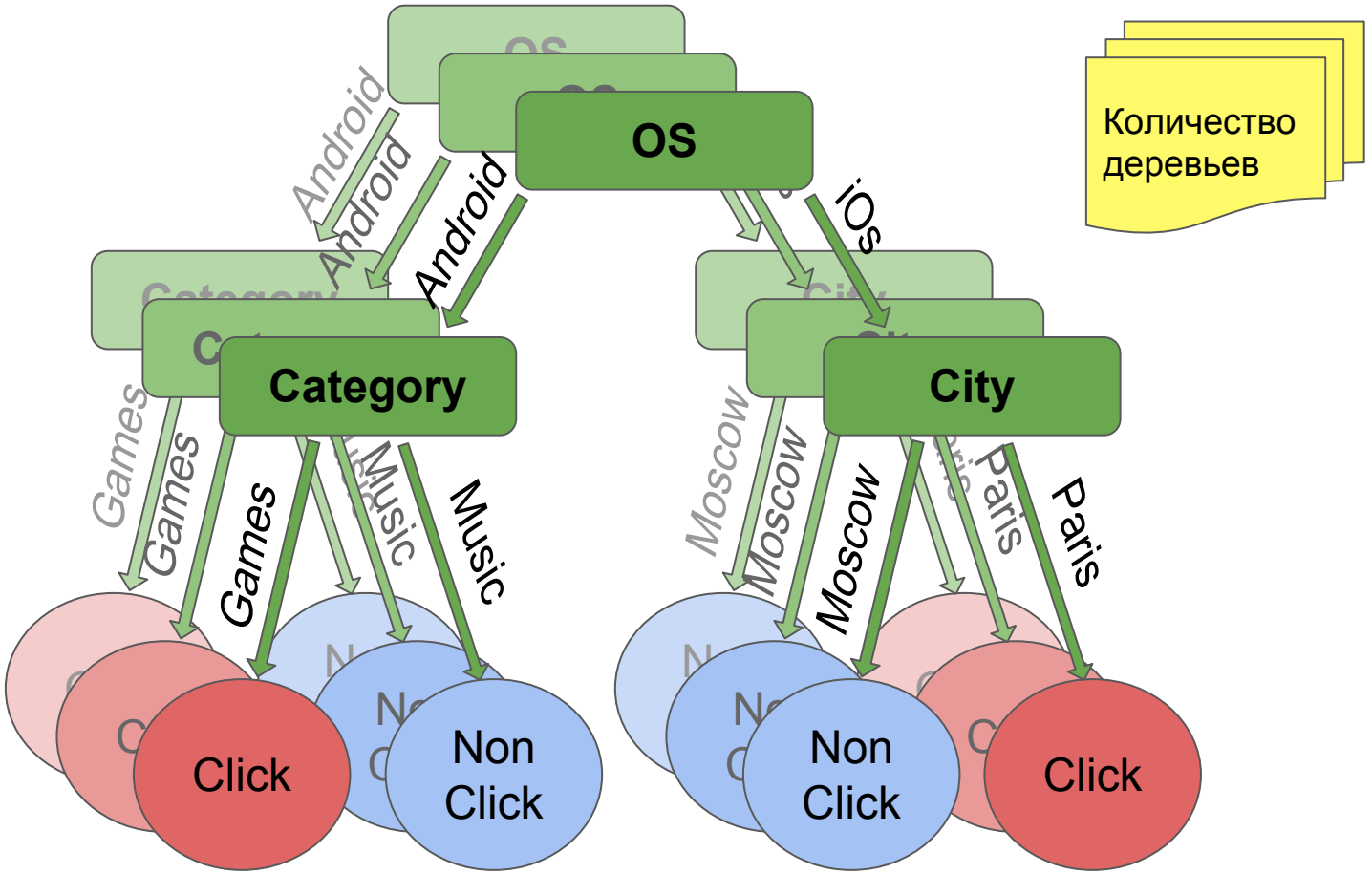


Алгоритм: **случайный лес** *(Random Forest)*

Дерево решений



Алгоритм: **случайный лес** *(Random Forest)*



Сырые данные

```
{
   "id":"951cb9f5-2bab-46ce-b759-8245cffxxxxx",
   "time":"2016-06-09T0:25:28Z",
   "bidfloor":2.88,
   "appOrSite":"app",
   "adType":"banner",
   "categories":"games,news,football",
   "publisherId":"11e281c1123139xxxxx",
   "carrier":"208-10",
   "os":"iOS",
   "connectionType":3,
   "coords":[48.929256439208984, 2.4255824089050293],
   "adSize":[320, 50],
   "exchange":"xxxxx",
   [...],
   "clicked":true
}
```

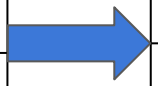
Os	MaxPrice	Time
Android	7.3	2016-06-09T0:25:28Z
iOS	4.55	2016-05-09T14:23:12Z
WindowsPhone	2.89	2016-06-09T11:35:11Z

Os	MaxPrice	Time
Android	7.3	2016-06-09T0:25:28Z
iOS	4.55	2016-05-09T14:23:12Z
WindowsPhone	2.89	2016-06-09T11:35:11Z

Os	MaxPrice	Time
Android	7.3	2016-06-09T0:25:28Z
iOS	4.55	2016-05-09T14:23:12Z
WindowsPhone	2.89	2016-06-09T11:35:11Z

Os	MaxPrice	Time
Android	7.3	2016-06-09T0:25:28Z
iOS	4.55	2016-05-09T14:23:12Z
WindowsPhone	2.89	2016-06-09T11:35:11Z

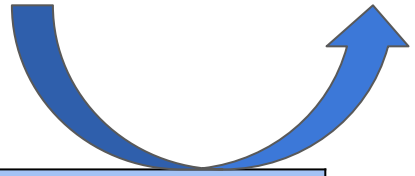
Os	MaxPrice	Time
Android	7.3	2016-06-09T0:25:28Z
iOS	4.55	2016-05-09T14:23:12Z
WindowsPhone	2.89	2016-06-09T11:35:11Z



Click
False
True
False

Os	MaxPrice	Time
Android	7.3	2016-06-09T0:25:28Z
iOS	4.55	2016-05-09T14:23:12Z
WindowsPhone	2.89	2016-06-09T11:35:11Z

Click
False
True
False

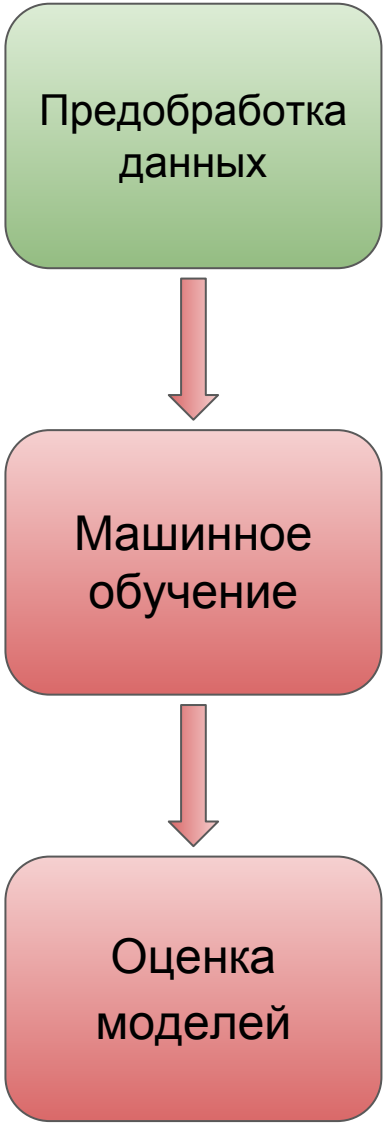


Os	MaxPrice	Time
3.0	6.0	1.0
5.0	3.0	5.0
1.0	2.0	3.0

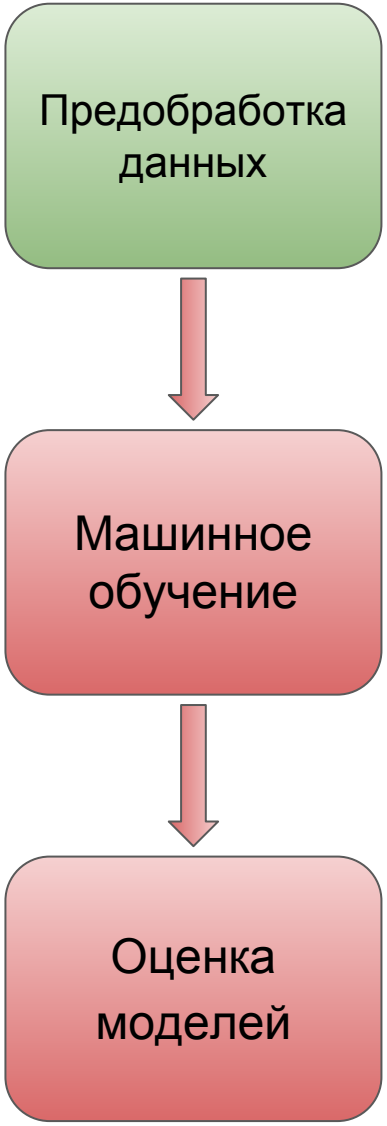
План.

1. **Почему Scala ?**
2. **Обзор библиотек Data-Science на Scala**
3. **Что за проблему решаем сегодня?**
4. **Замарать руки в коде**
5. **Подвести итоги и выбрать**

подходящую нам библиотеку!


Предобработка данных
Машинное обучение
Оценка моделей

Saddle



Saddle

Создаем датафрейм

```
val dataframe: Frame[Int, String, Any] =
  Frame(requestsParsed, columns)
    .rdropNA
```

Saddle

Создаем датафрейм

```
val dataframe: Frame[Int, String, Any] =
  Frame(requestsParsed, columns)
    .rdropNA
```

Saddle

Создаем датафрейм

```
val dataframe: Frame[Int, String, Any] =
  Frame(requestsParsed, columns)
    .rdropNA
```

Saddle

Создаем датафрейм

```
val dataframe: Frame[Int, String, Any] =
  Frame(requestsParsed, columns)
    .rdropNA
              .rmap              .rsqueeze

              .rconcat           .rmask

              .rfilter           .rtransform

              .rjoin             .rwhere
```

Saddle

Создаем датафрейм

```
val dataframe: Frame[Int, String, Any] =
  Frame(requestsParsed, columns)
    .rdropNA
```

row-wise

.rmap .rsqueeze

.rconcat .rmask

.rfilter .rtransform

.rjoin .rwhere

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)

val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.

val transformed                                    ble] = dataframe
  .col("city")
  .rtransform {
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)

val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)

val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Frame[Int, String, Any] => Seq[Any]

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)
```

List((Paris,0), (London,1), (Moscow,2), (Cherkessk,3)

```
val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)
```

Map(London -> 1.0, Moscow -> 2.0, Paris -> 0.0, Cherkessk -> 3.0)

```
val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)
```

Map(London -> 1.0, Moscow -> 2.0, Paris -> 0.0, Cherkessk -> 3.0)

```
val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)

val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)

val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndexed: Map[String, Double] = dataframe
  .col("city")
  .toSeq
  .map(_._3)
  .distinct
  .zipWithIndex
  .toMap
  .mapValues(_.toDouble)
```

Series[String, Any]

```
val transformedDataframe: Frame[Int, String, Double] = dataframe
  .col("city")
  .rtransform { series =>
    val citySeries = "city" -> cityIndexed(
      series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Индексируем категориальные данные

```
val cityIndex
  .col("city"
  .toSeq
  .map(_._3)
  .distinct
  .zipWithInd
  .toMap
  .mapValues(

val transform                                          = dataframe
  .col("city"
  .rtransform
    val cityS
```

```
series.get("city").get.asInstanceOf[String]
    )
    series.concat(citySeries)
  }
```

Saddle

Разбиваем данные случайным образом на тренировочную и тестовую выборки

```
val shuffledDF: Frame[Int, String, Double] =
  transformedDataframe
    .sortedRowsBy(_ => Random.nextDouble)
    .resetRowIndex

 val (testSet, trainSet) = shuffledDF
   .rowSplitAt((0.1 * shuffledDF.numRows).toInt)
```

Saddle

Разбиваем данные случайным образом на тренировочную и тестовую выборки

```
val shuffledDF: Frame[Int, String, Double] =
  transformedDataframe
    .sortedRowsBy(_ => Random.nextDouble)
    .resetRowIndex

 val (testSet, trainSet) = shuffledDF
   .rowSplitAt((0.1 * shuffledDF.numRows).toInt)
```

Saddle

Разбиваем данные случайным образом на тренировочную и тестовую выборки

```
val shuffledDF: Frame[Int, String, Double] =
  transformedDataframe
    .sortedRowsBy(_ => Random.nextDouble)
    .resetRowIndex

 val (testSet, trainSet) = shuffledDF
   .rowSplitAt((0.1 * shuffledDF.numRows).toInt)
```

Saddle

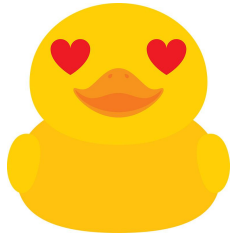
Разбиваем данные случайным образом на тренировочную и тестовую выборки

```
val shuffledDF: Frame[Int, String, Double] =
  transformedDataframe
    .sortedRowsBy(_ => Random.nextDouble)
    .resetRowIndex

 val (testSet, trainSet) = shuffledDF
   .rowSplitAt((0.1 * shuffledDF.numRows).toInt)
```

Saddle

Удобные для науки о данных структуры :

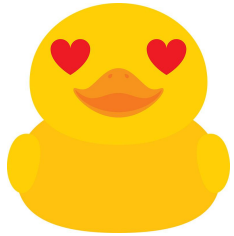
фреймы, матрицы, серии, векторы



Saddle

Удобные для науки о данных структуры :

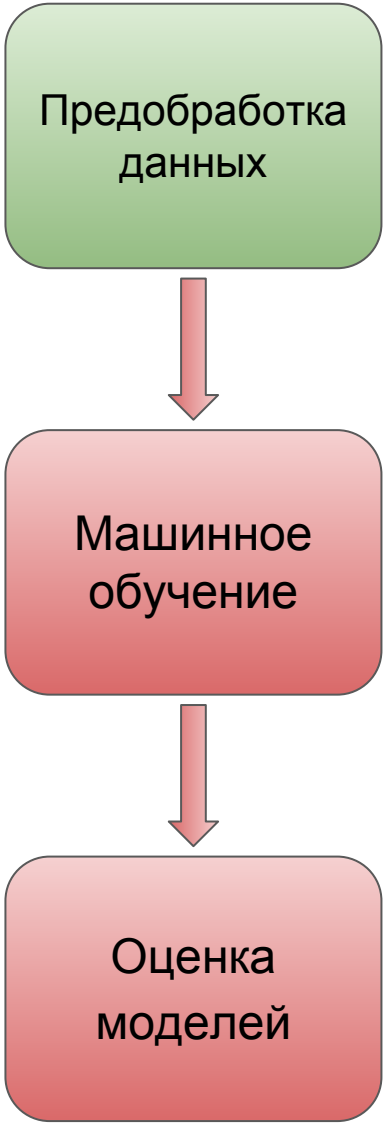
фреймы, матрицы, серии, векторы



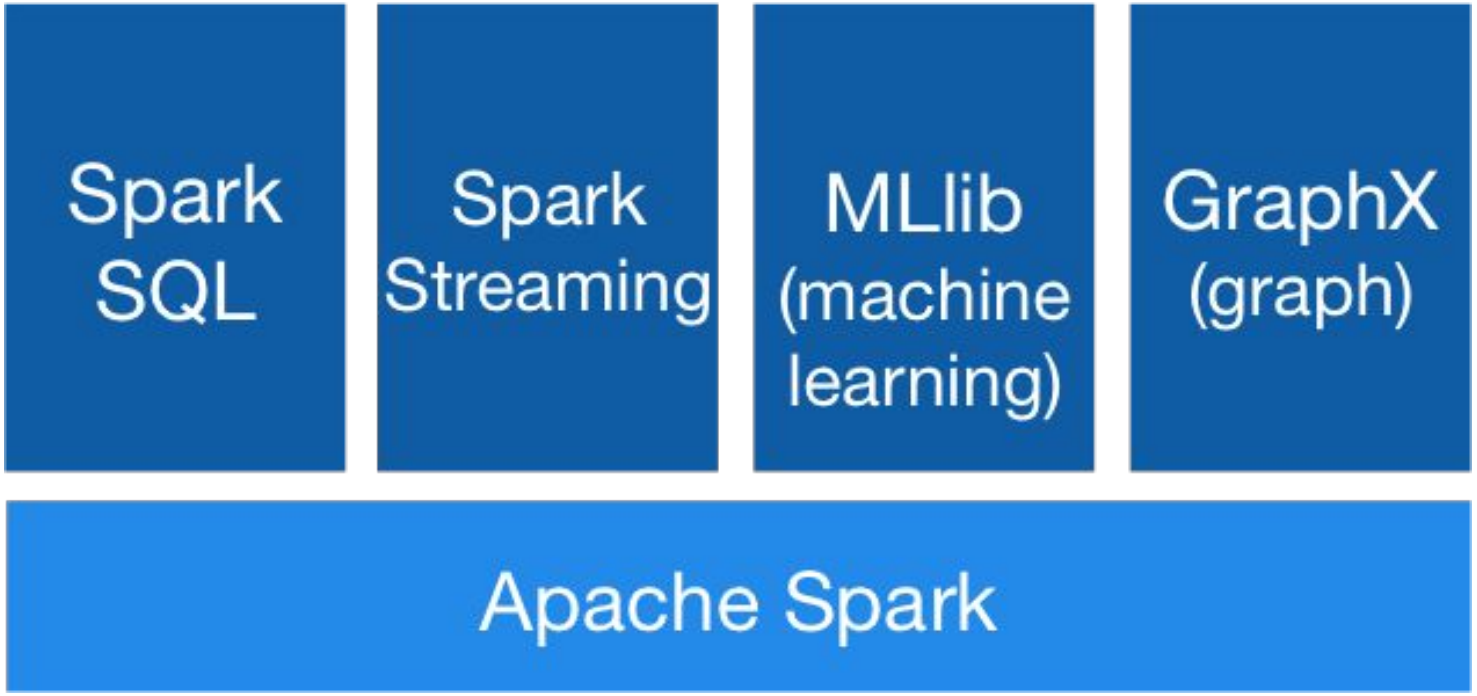
Не в активной разработке

Не типизированные датафреймы

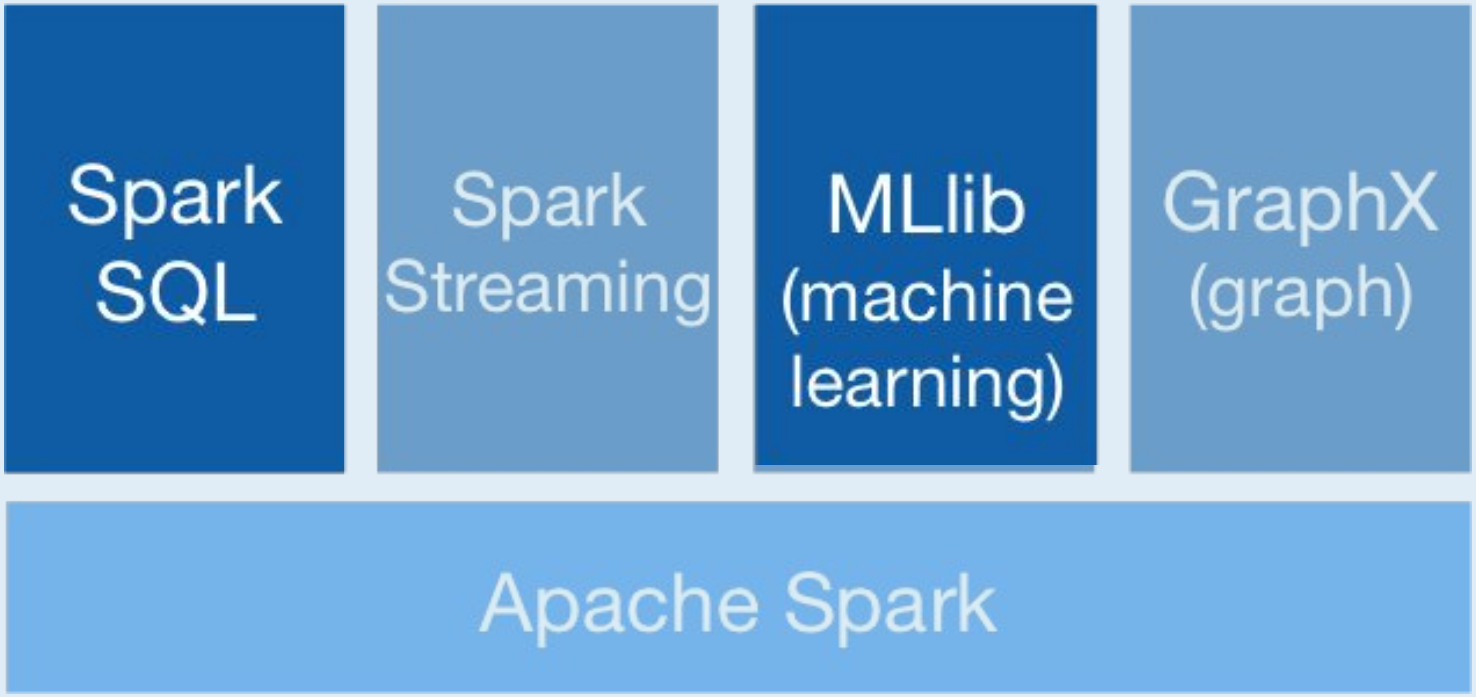
Spark



Spark


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

Spark


Spark SQL
Spark Streaming
MLlib (machine learning)
GraphX (graph)
Apache Spark

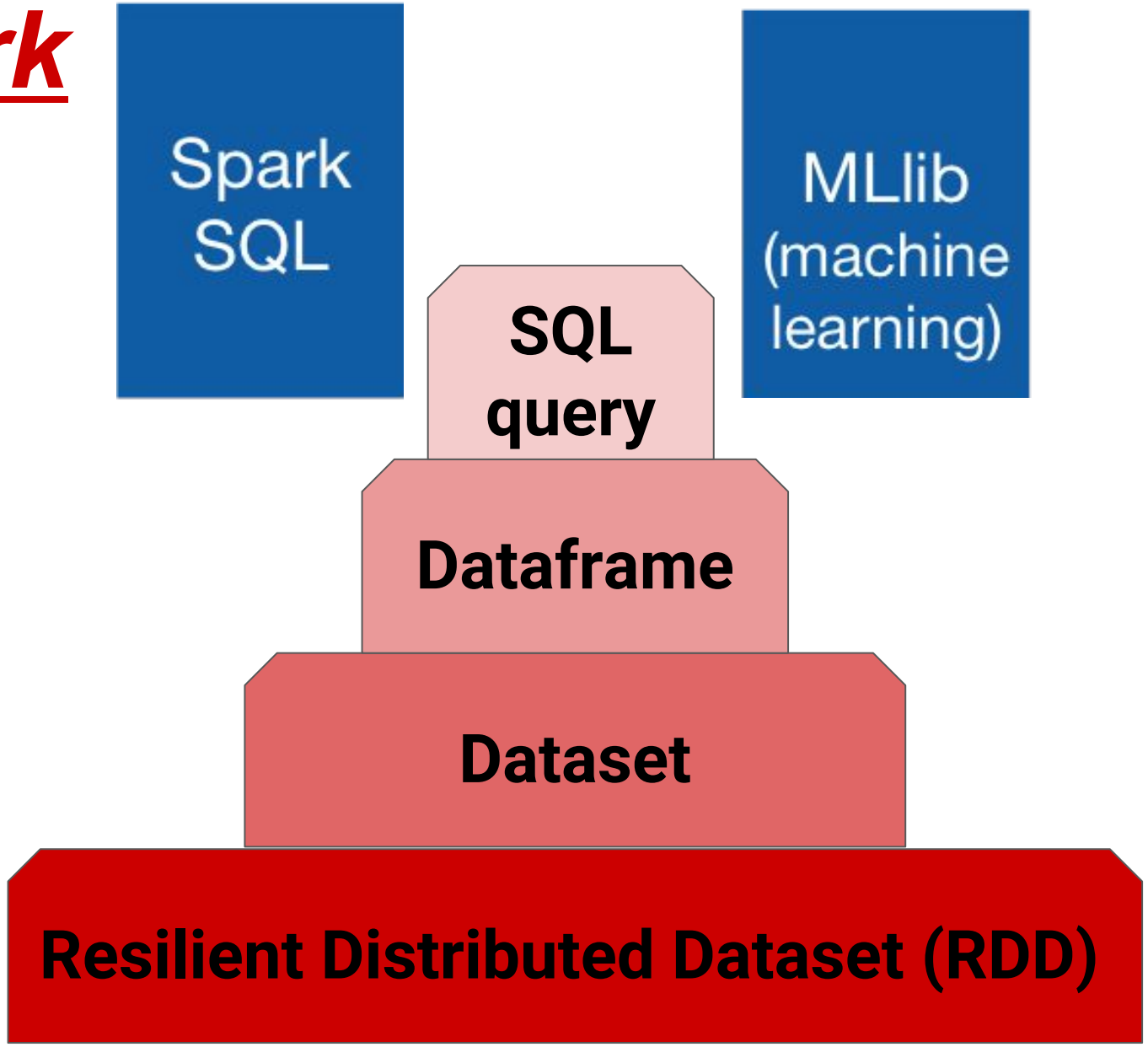
Spark

Spark

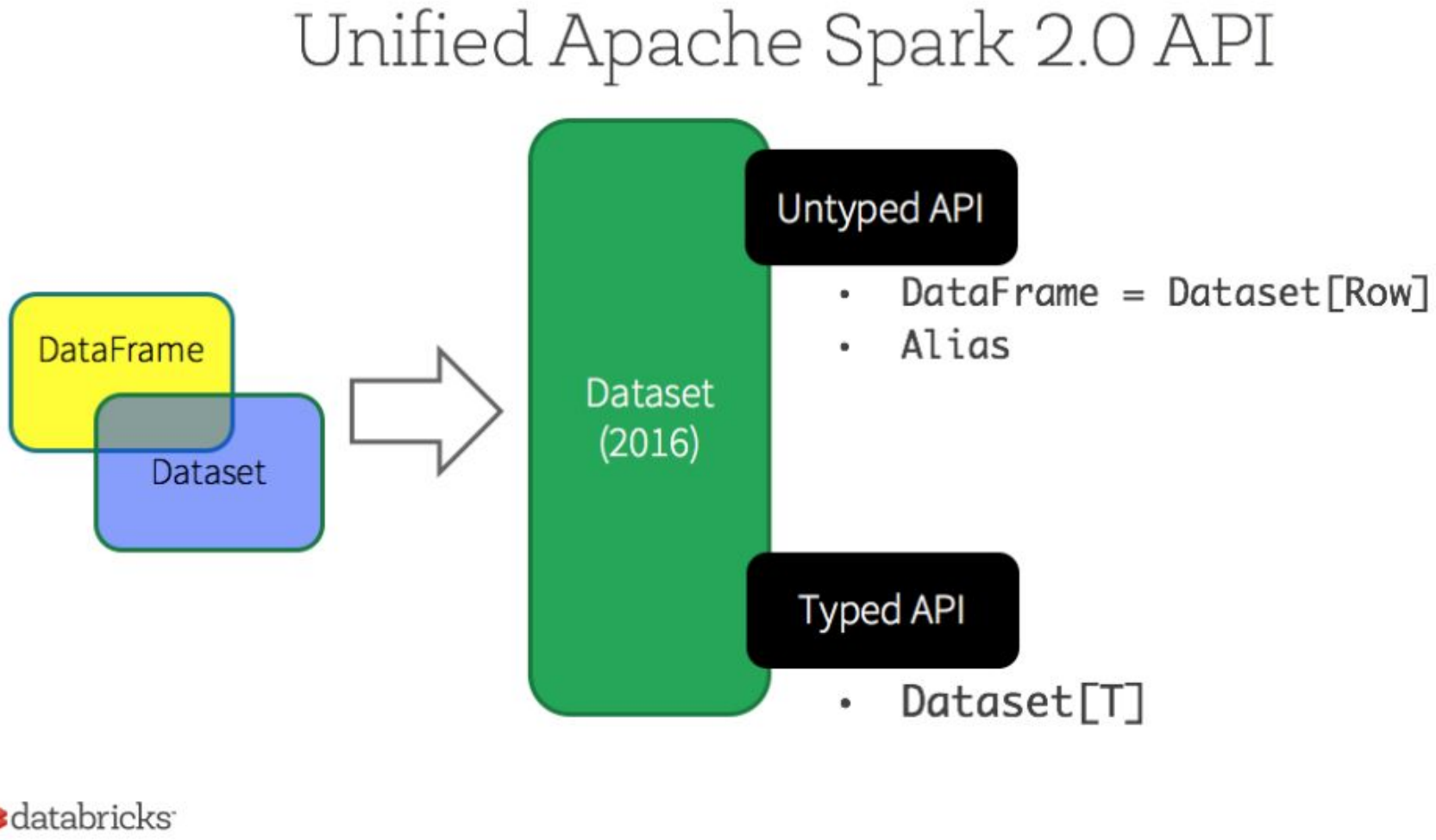
Spark

Resilient Distributed Dataset (RDD)

Spark


Spark SQL
MLlib (machine learning)
SQL query
Dataframe
Dataset
Resilient Distributed Dataset (RDD)

Spark


Unified Apache Spark 2.0 API
DataFrame
Dataset
Dataset
(2016)
Untyped API
• DataFrame = Dataset[Row]
• Alias
Typed API
• Dataset[T]
databricks

Создаем датафрейм

```
val rawDF: DataFrame = sparkSession
  .read
  .json(inputFile)
```

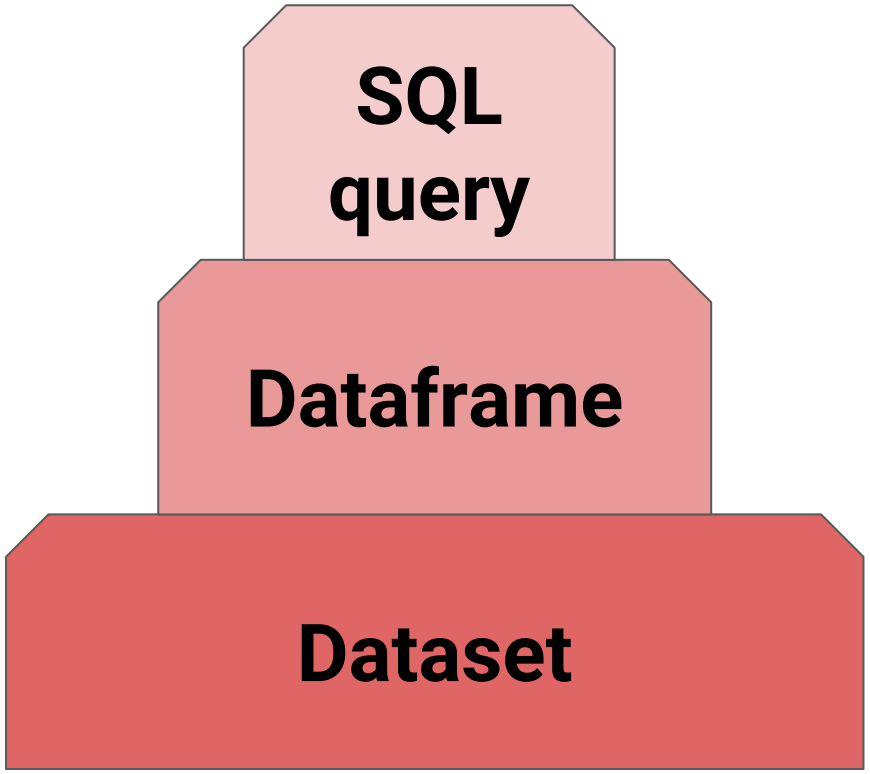
Spark

Создаем датафрейм

```
val rawDF: DataFrame = sparkSession
  .read
  .json(inputFile)
```

Spark

	SQL	DataFrames	Datasets
Syntax Errors	Runtime	Compile Time	Compile Time
Analysis Errors	Runtime	Runtime	Compile Time



Spark

SQL query

```
dataframe.createOrReplaceTempView("data")

sparkSession.sql("
  SELECT * FROM data WHERE click=true
")
```

Spark

SQL query

```
dataframe.createOrReplaceTempView("data")

sparkSession.sql("
  SELECT * FROM data WHERE click=true
")
```

Spark

SQL query

```
dataframe.createOrReplaceTempView("data")

sparkSession.sql("
  SELECT * FROM data WHERE click=true
")
```

Spark

SQL query

```
dataframe.createOrReplaceTempView("data")

sparkSession.sql("
  SELECT * FROM data WHERE click=true
")
```

Spark

SQL query

```
dataframe.createOrReplaceTempView("data")

sparkSession.sql("
  SELECT * FROM data WHERE click=true
")
```

Spark

SQL query

```
dataframe.createOrReplaceTempView("data")

sparkSession.sql("
  WRITE WHAT U WANT CRAZY_OPERATOR CRAZY_COLUMN=true
")
```

Dataframe

```
dataframe.where($"click" === true)
```

Dataframe

```
dataframe.where($"click" === true)
```

Spark

Dataframe

```
dataframe.where($"CRAZY_COLUMN" === true)
```

Dataframe

```
dataframe.CRAZY_METHOD($"click" === true)
```

Spark

Dataframe

```
dataframe.count

      .join                .sort

      .filter              .where

      .groupBy             .select

      .agg                 .union

      .orderBy             .drop
```

Создаём датасет

Dataset

```
case class Request(
  os: String,
  carrier: String,
  timestamp: Double,
  city: String,
  ...
  clicked: Boolean
)
```

```
val dataset: Dataset[Request] = dataframe
  .select("timestamp", "city", "clicked", …, "os")
  .as[Request]
```

Создаём датасет

Dataset

```
case class Request(
  os: String,
  carrier: String,
  timestamp: Double,
  city: String,
  ...
  clicked: Boolean
)
```

```
val dataset: Dataset[Request] = dataframe
  .select("timestamp", "city", "clicked", …, "os")
  .as[Request]
```

Создаём датасет

Dataset

```
case class Request(
  os: String,
  carrier: String,
  timestamp: Double,
  city: String,
  ...
  clicked: Boolean
)
```

```
val dataset: Dataset[Request] = dataframe
  .select("timestamp", "city", "clicked", …, "os")
  .as[Request]
```

Создаём датасет

Dataset

```
case class Request(
  cs: String,
  carrier: String,
  timestamp: Double,
  city: String,
  ...
  Clicked: Boolean
)
```

Dataset[Row]

```
val dataset: Dataset[Request] = dataframe
  .select("timestamp", "city", "clicked", …, "os")
  .as[Request]
```

Dataset

```
dataset.filter(_.click==true)
```

Spark

Dataset

```
dataset.CRAZY_OPERATOR(_.CRAZY_COLUMN==true)
```

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

Estimator

Transformer

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

Estimator

Transformer

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

Estimator

Transformer

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Индексируем категориальные данные

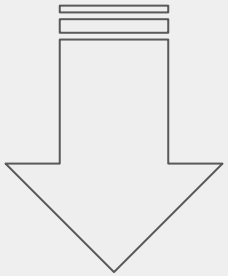
```
val indexer = new StringIndexer()
  .setInputCol("city")
  .setOutputCol("cityIdx")
  .fit(dataframe)

val indexedDataframe: DataFrame = indexer
  .transform(dataframe)
```

City	Index
Saint Petersburg	0
Montpellier	1
Moscow	2

Адаптируем данные к формату ожидаемому на входе алгоритма

Dataframe

os	city	category	position	publisher	click
1.0	11.0	15.0	1.0	16.0	1.0
7.0	789.0	25.0	0.0	271.0	0.0



features	label
[1.0, 11.0, 15.0, 1.0, 16.0]	1.0
[7.0, 789.0, 25.0, 0.0, 271.0, 0.0]	0.0

Адаптируем данные к формату ожидаемому на входе алгоритма

Dataframe

os	city	category	position	publisher	click
1.0	11.0	15.0	1.0	16.0	1.0
7.0	789.0	25.0	0.0	271.0	0.0

```
val assembler: VectorAssembler = new VectorAssembler()
  .setInputCols(featuresList)
  .setOutputCol("features")

val finalDataframe: DataFrame = assembler
  .transform(dataframe)
```

features	label
[1.0, 11.0, 15.0, 1.0, 16.0]	1.0
[7.0, 789.0, 25.0, 0.0, 271.0, 0.0]	0.0

Адаптируем данные к формату ожидаемому на входе алгоритма

Dataframe

os	city	category	position	publisher	click
1.0	11.0	15.0	1.0	16.0	1.0
7.0	789.0	25.0	0.0	271.0	0.0

```
val assembler: VectorAssembler = new VectorAssembler()
  .setInputCols(featuresList)
  .setOutputCol("features")

val finalDataframe: DataFrame = assembler
  .transform(dataframe)
```

features	label
[1.0, 11.0, 15.0, 1.0, 16.0]	1.0
[7.0, 789.0, 25.0, 0.0, 271.0, 0.0]	0.0

Разбиваем данные случайным образом на тренировочную и тестовую выборки

Dataframe

```
val Array(trainSet, testSet) =
  finalDataframe
    .randomSplit(Array(0.9, 0.1))
```

Spark

Spark

- **Готовые оптимизированные методы для feature engineering/selection**

Spark

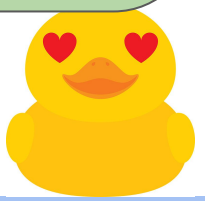
- **Готовые оптимизированные методы для feature engineering/selection**
- **Типизированные структуры - Datasets**

Spark

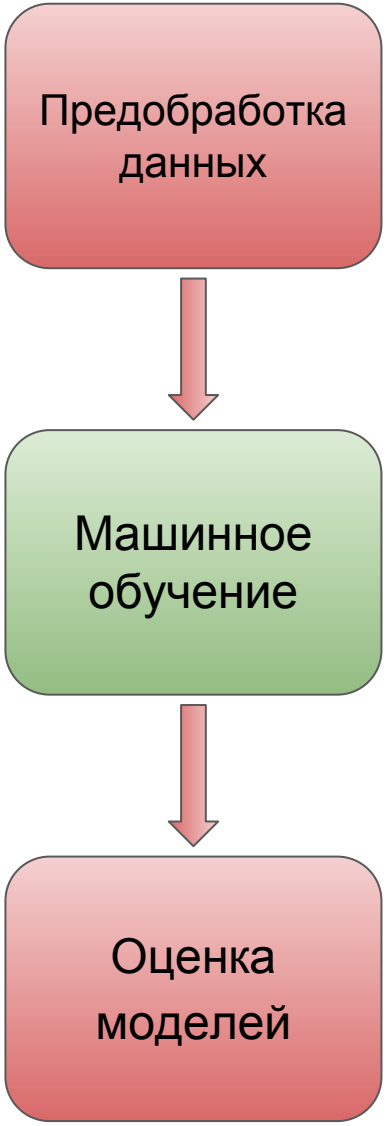
- **Готовые оптимизированные методы для feature engineering/selection**
- **Типизированные структуры - Datasets**
- **Возможность очень типизированных структур с Frameless Datasets**

Spark

- **Готовые оптимизированные методы для feature engineering/selection**
- **Типизированные структуры - Datasets**
- **Возможность очень типизированных структур с Frameless Datasets**
- **Возможность делать SQL с минимальным уровнем типизации : синтаксиса**



Spark



Конструируем классификатор и конфигурируем его

```
val forest = new RandomForestClassifier()
  .setLabelCol("click")
  .setFeaturesCol("features")
  .setNumTrees(100)
  .setMaxDepth(30)
```

Spark

Конструируем классификатор и конфигурируем его

```
val forest = new RandomForestClassifier()
  .setLabelCol("click")
  .setFeaturesCol("features")
  .setNumTrees(100)
  .setMaxDepth(30)
```

```
.setMaxBins
.setCacheNodeIds
.setSeed
.setThresholds
.setImpurity
```

```
.setMinInfoGain
.setCheckpointInterval
.setMaxMemoryInMB
.setMinInstancesPerNode
.setFeatureSubsetStrategy
```

Spark

Запускаем обучение на обучающей выборке

```
val model: RandomForestClassificationModel =
  forest
    .fit(trainSet)
```

```
model
.write                          .impurity
.treeWeights                    .checkpointInterval
.featureImportances             .getMinInfoGain
._trees                         .getFeatureSubsetStrategy
```

```
val prediction: DataFrame =
  model
    .transform(testSet)
```

Spark

Запускаем обучение на обучающей выборке

```
val model: RandomForestClassificationModel =
  forest
    .fit(trainSet)
```

```
model
.write                          .impurity
.treeWeights                    .checkpointInterval
.featureImportances             .getMinInfoGain
._trees                         .getFeatureSubsetStrategy
```

```
val prediction: DataFrame =
  model
    .transform(testSet)
```

Запускаем обучение на обучающей выборке

```
val model: RandomForestClassificationModel =
  forest
    .fit(trainSet)
```

```
model
.write                          .impurity
.treeWeights                    .checkpointInterval
.featureImportances             .getMinInfoGain
._trees                         .getFeatureSubsetStrategy
```

```
val prediction: DataFrame =
  model
    .transform(testSet)
```

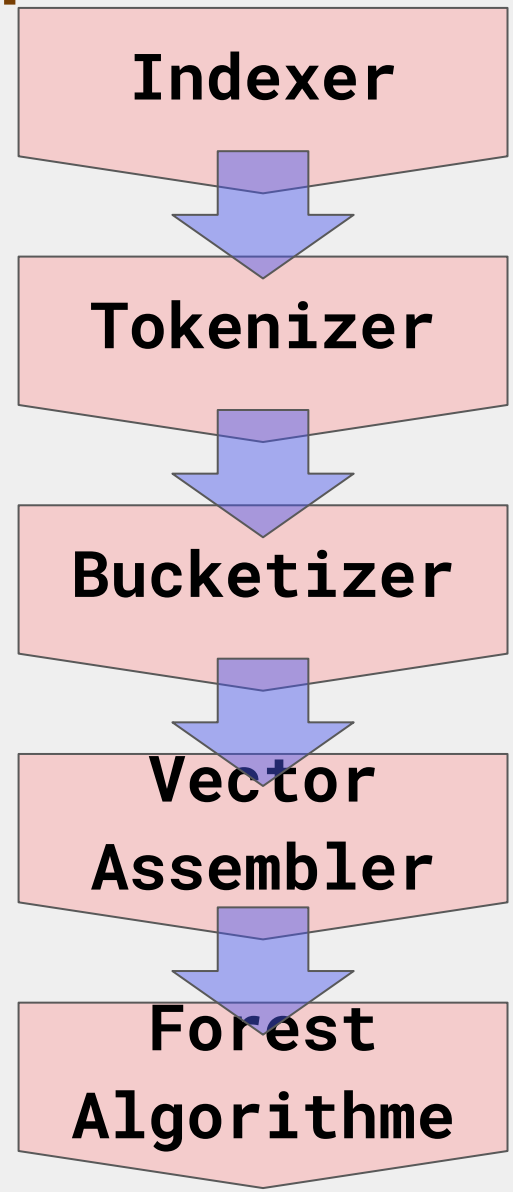
Запускаем обучение на обучающей выборке

```
val model: RandomForestClassificationModel =
  forest
    .fit(trainSet)
```

```
model
.write                          .impurity
.treeWeights                    .checkpointInterval
.featureImportances             .getMinInfoGain
._trees                         .getFeatureSubsetStrategy
```

```
val prediction: DataFrame =
  model
    .transform(testSet)
```

Собираем все элементы в Pipeline



Собираем все элементы в Pipeline

```
val randomForestPipeline = new Pipeline()
  .setStages(Array(
     indexer,
     tokenizer,
     bucketizer,
     vectorAssembler,
     forest)
  )
```

```
val model = randomForestPipeline.fit(trainSet)
```

Собираем все элементы в Pipeline

```
val randomForestPipeline = new Pipeline()
  .setStages(Array(
     indexer,
     tokenizer,
     bucketizer,
     vectorAssembler,
     forest)
  )

val model = randomForestPipeline.fit(trainSet)
```

Spark Собираем все элементы в Pipeline

```
val randomForestPipeline = new Pipeline()
  .setStages(Array(
    indexer,
     tokenizer,
     bucketizer,
     vectorAssembler,
     forest)
  )

val model = randomForestPipeline.fit(trainSet)
```

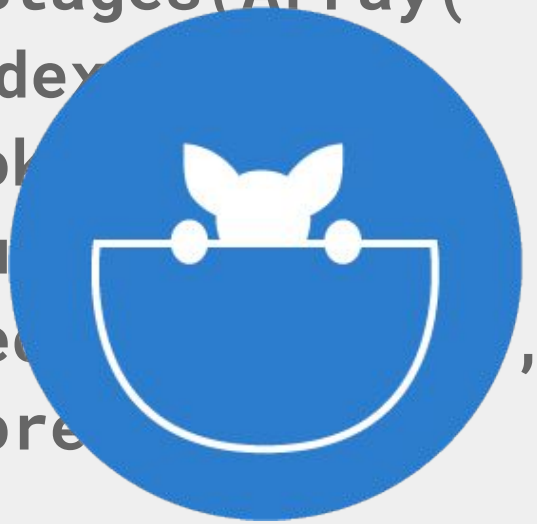
Spark Собираем все элементы в Pipeline

```
val randomForestPipeline = new Pipeline()
  .setStages(Array(
    indexer,
     tokenizer,
     bucketizer,
     vectorAssembler,
     forest)
  )
```

```
val model = randomForestPipeline.fit(trainSet)
```

Spark Собираем все элементы в Pipeline

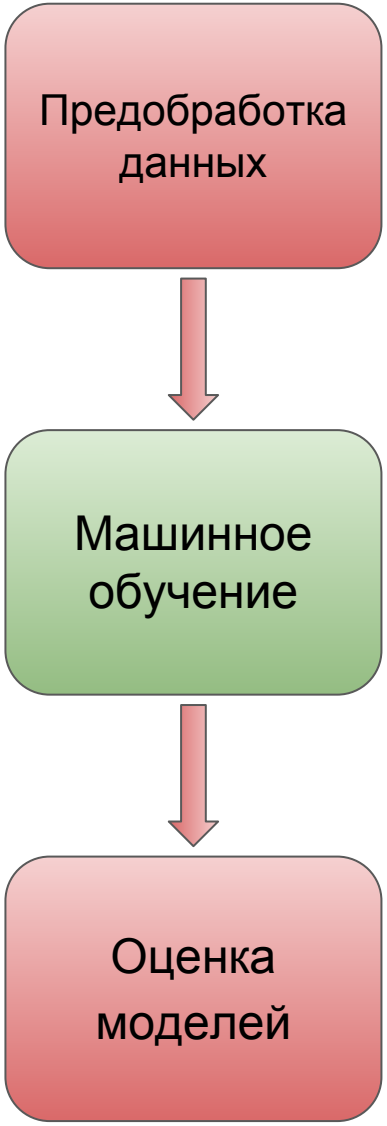
```
val randomForestPipeline = new Pipeline()
  .setStages(Array(
    index
    tok
    bu
    ve
    fore
  ))
```



mleap

```
val model = randomForestPipeline.fit(trainSet)
```

Smile



Smile

Конструируем классификатор и конфигурируем его

```
val model = new RandomForest(
  x = featuresTrainArray,
  y = labelTrainArray,
  ntrees = numberOfTrees,
  maxNodes = 100,
  mtry = mtry,
  subsample = 1.0,
  rule = DecisionTree.SplitRule.GINI,
  classWeight = (1, 10)
)
```

Запускаем классификатор на обучающей выборке

Smile

Конструируем классификатор и конфигурируем его

```
val model = new RandomForest(
  x = featuresTrainArray,
  y = labelTrainArray,
  attributes = nominalAttributes,
  ntrees = numberOfTrees,
  maxNodes = 100,
  mtry = mtry,
  subsample = 1.0,
  rule = DecisionTree.SplitRule.GINI,
  classWeight = (1, 2)
)
```

Запускаем классификатор на обучающей выборке

Smile

Конструируем классификатор и конфигурируем его

```
val model = new RandomForest(
  x = featuresTrainArray,
  y = labelTrainArray,
  attributes = nominalAttributes,
  ntrees = numberOfTrees,
  maxNodes = 100,
  mtry = mtry,
  subsample = 1.0,
  rule = DecisionTree.SplitRule.GINI,
  classWeight = (1, 2)
)
```

Запускаем классификатор на обучающей выборке

Smile

Делаем предсказания на тестовой выборке

```
val prediction = model.predict(features)
```

```
.trim          .getTrees
.size          .error
.test          .importance
```

```
model.importance
```

Smile

Делаем предсказания на тестовой выборке

```
val prediction = model.predict(features)
```

```
.trim          .getTrees
.size          .error
.test          .importance
```

```
model.importance
```

Smile

Делаем предсказания на тестовой выборке

```
val prediction = model.predict(features)
```

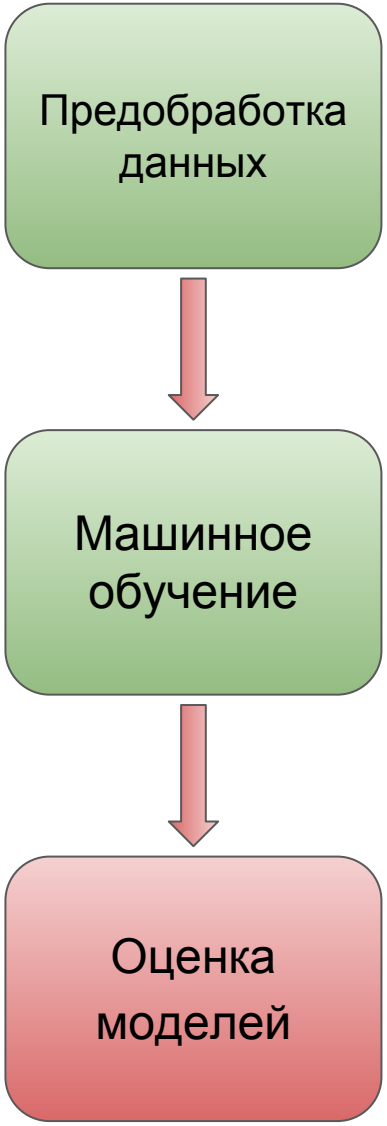
```
.trim          .getTrees
.size          .error
.test          .importance
```

```
model.importance
```

```
0.17041644829479835,0.0,0.2461154091553050 5,1.1389295846602683,0.07655364222
388063,0.0,0.0,0.009896625232551026,4.57453119760533,0.3604788069073785 5,1.2
020833333333334,0.007662298205433167,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0
```

Spark

Smile



Spark

Метрики оценки моделей

Smile

Regression

Binary Classification

Multiclass Classification

Regression

Classification

Spark

Метрики оценки моделей

Smile

Regression

Binary Classification

Multiclass Classification

Regression

Classification

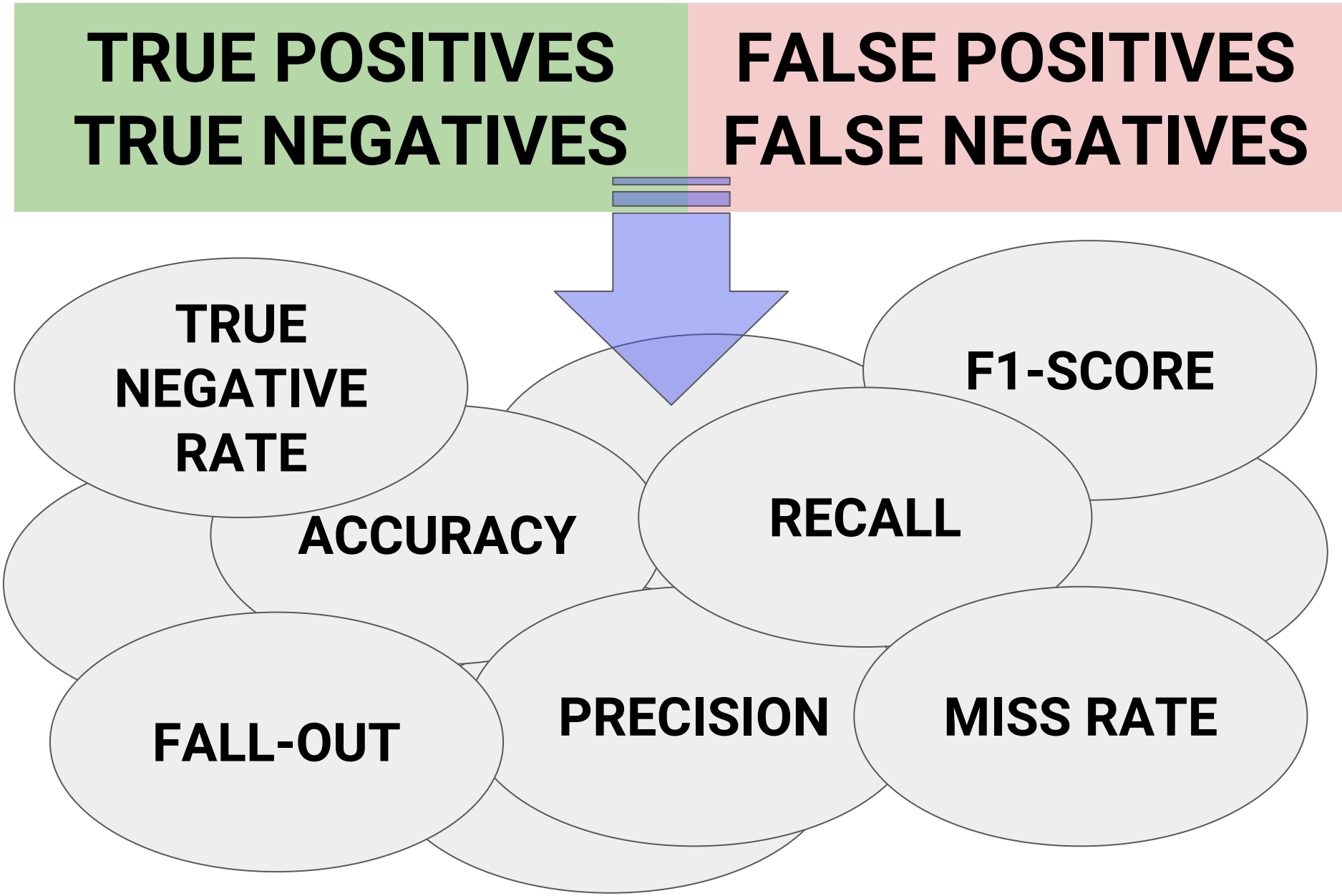
	Был клик	Не было клика
Предсказание клика	**100**	**20**
Предсказание нет клика	**300**	**580**

FALSE POSITIVES
FALSE NEGATIVES

	Был клик	Не было клика
Предсказание клика	100	20
Предсказание нет клика	300	580

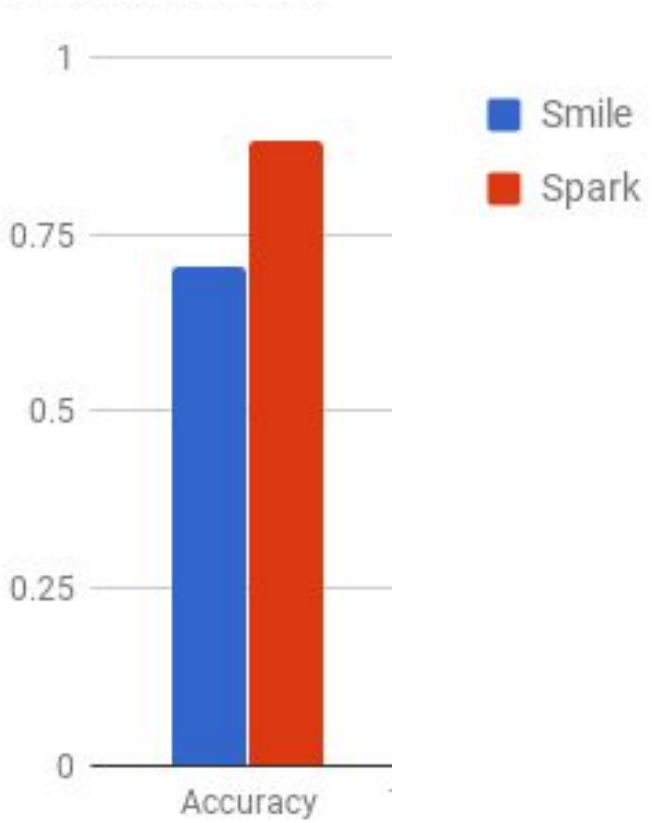
TRUE POSITIVES
TRUE NEGATIVES

	Был клик	Не было клика
Предсказание клика	100	20
Предсказание нет клика	300	580


TRUE POSITIVES
TRUE NEGATIVES
FALSE POSITIVES
FALSE NEGATIVES
TRUE NEGATIVE RATE
F1-SCORE
ACCURACY
RECALL
FALL-OUT
PRECISION
MISS RATE

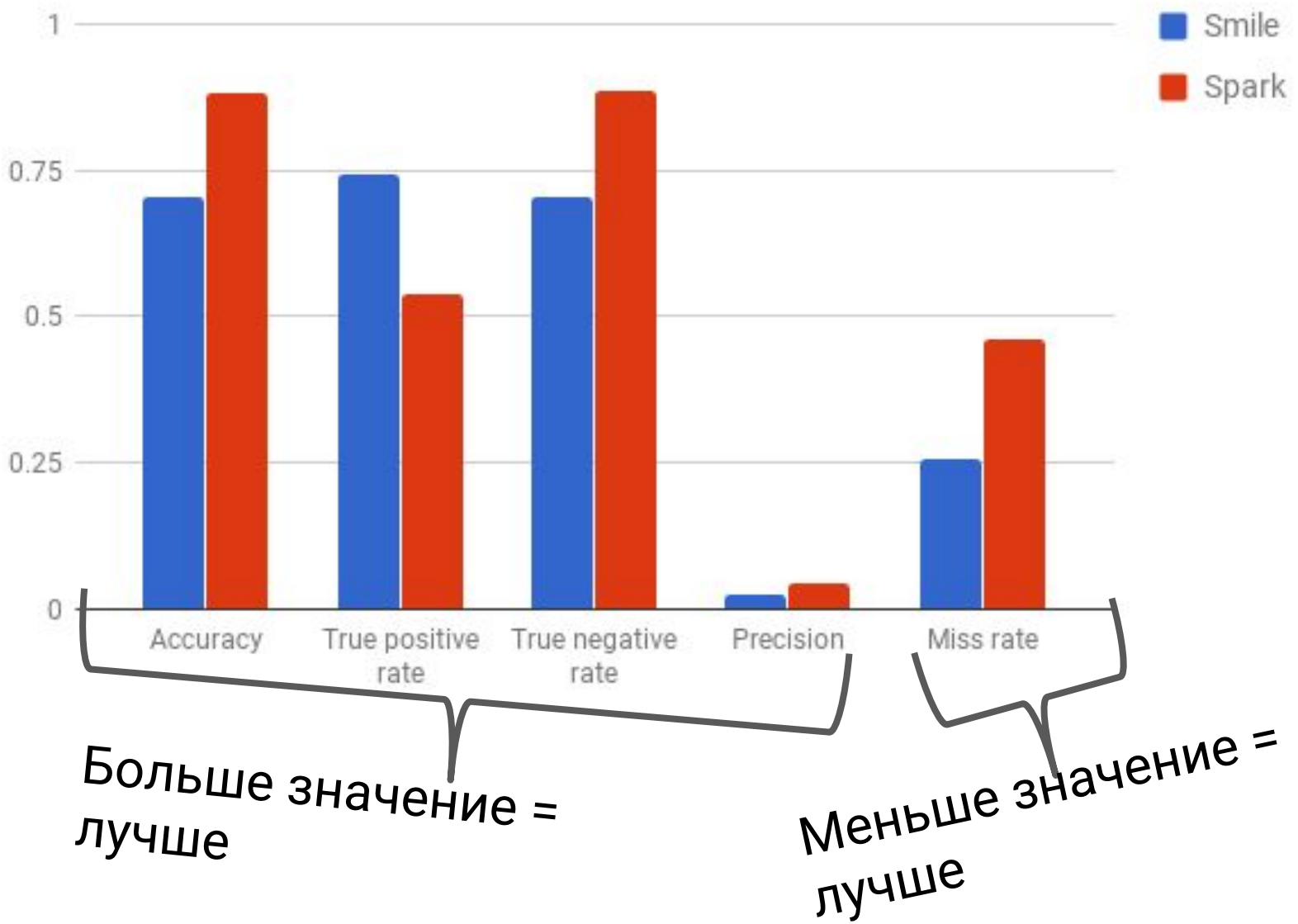
Сравниваем Случайный лес Spark и Smile

Метрики классификации



Сравниваем Случайный лес Spark и Smile

Метрики классификации



Подбираем оптимальные параметры алгоритма

```
val forest = new RandomForestClassifier()
  .setLabelCol("click")
  .setFeaturesCol("features")

val randomForestGrid = new ParamGridBuilder()
  .addGrid(forest.impurity, Array("entropy", "gini"))
  .addGrid(forest.cacheNodeIds, Array(true, false))
  .addGrid(forest.maxDepth, Array(5, 10, 20, 30))
  .addGrid(forest.maxMemoryInMB, Array(256, 512) )
  .build()
```

Подбираем оптимальные параметры алгоритма

```
val forest = new RandomForestClassifier()
  .setLabelCol("click")
  .setFeaturesCol("features")

val randomForestGrid = new ParamGridBuilder()
  .addGrid(forest.impurity, Array("entropy", "gini"))
  .addGrid(forest.cacheNodeIds, Array(true, false))
  .addGrid(forest.maxDepth, Array(5, 10, 20, 30))
  .addGrid(forest.maxMemoryInMB, Array(256, 512) )
  .build()
```

Spark

Подбираем оптимальные параметры алгоритма

```
val trainValidationSplit =
   new TrainValidationSplit()
     .setEstimator(tree)
     .setEvaluator(new BinaryClassificationEvaluator)
     .setEstimatorParamMaps(randomForestGrid)
     .setTrainRatio(0.7)


val bestModel = trainValidationSplit.fit(trainSet)
```

Подбираем оптимальные параметры алгоритма

```
val trainValidationSplit =
   new TrainValidationSplit()
     .setEstimator(tree)
     .setEvaluator(new BinaryClassificationEvaluator)
     .setEstimatorParamMaps(randomForestGrid)
     .setTrainRatio(0.7)


val bestModel = trainValidationSplit.fit(trainSet)
```

Spark

Подбираем оптимальные параметры алгоритма

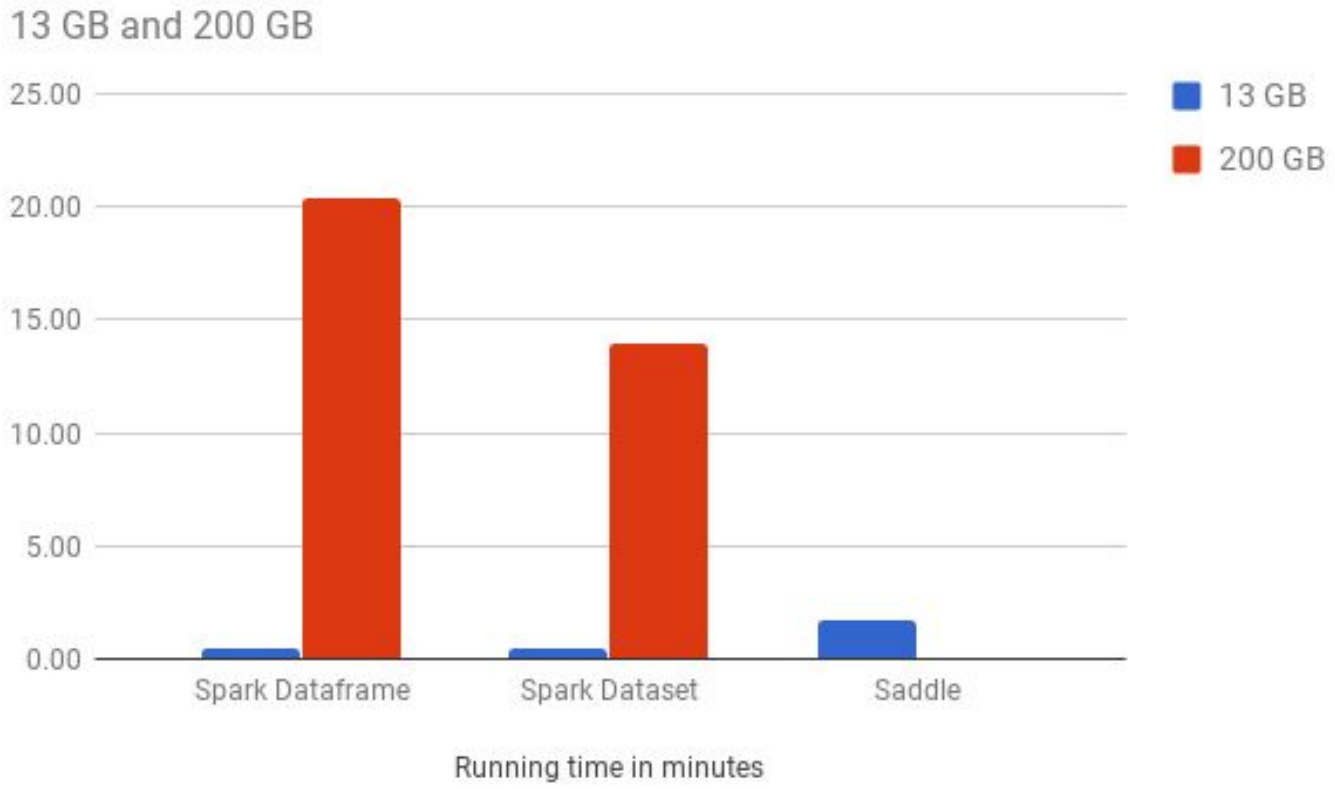
```
val trainValidationSplit =
   new TrainValidationSplit()
     .setEstimator(tree)
     .setEvaluator(new BinaryClassificationEvaluator)
     .setEstimatorParamMaps(randomForestGrid)
     .setTrainRatio(0.7)


val bestModel = trainValidationSplit.fit(trainSet)
```

План.

1. Почему Scala ?
2. Обзор библиотек Data-Science на Scala
3. Что за проблему решаем сегодня?
4. Замарать руки в коде
5. Подвести итоги и выбрать

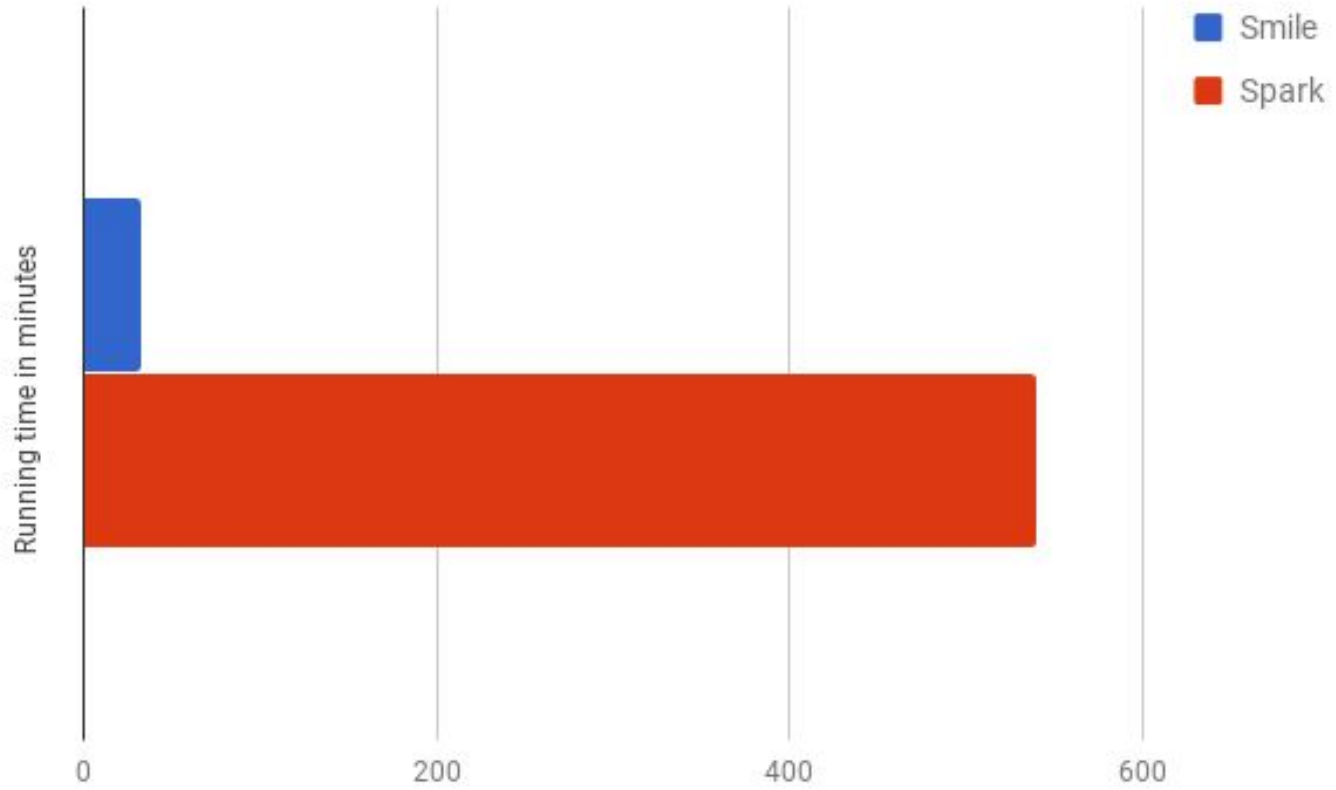
подходящую нам библиотеку!

Предобработка данных Spark и Saddle



64 GB RAM, 16 CPU

Обучение : Случайный лес Spark и Smile на 200 GB данных



Smile : 64 GB RAM, 16 CPU

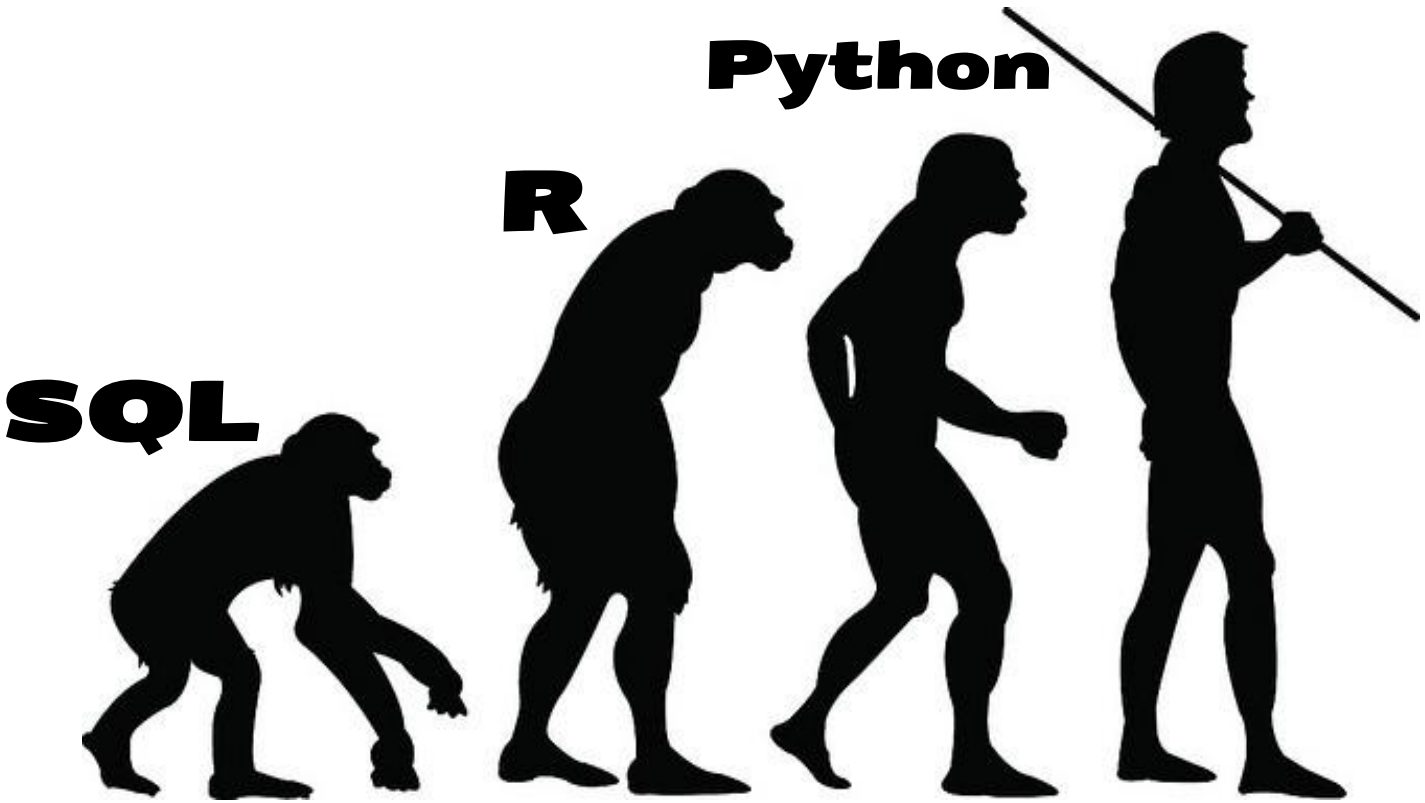
Spark : cluster 7 nodes, each of 30 GB RAM/16 CPU

Мой List[tools]

(для сегодняшнего примера!!):

Preprocessing	**Spark**
Machine Learning (**Random Forest**)	**Smile**

SCALABLE :
GROWABLE & ENABLING GROWTH



SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

- **Фокус на решение задач в контексте бизнеса**
- **Фокус на решение задач в контексте науки о данных**

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

- **Фокус на решение задач в контексте бизнеса**
- **Фокус на решение задач в контексте науки о данных**
- **Уже сейчас : богатый функционал библиотек….**

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

- **Фокус на решение задач в контексте бизнеса**
- **Фокус на решение задач в контексте науки о данных**
- **Уже сейчас : богатый функционал библиотек….**
- **и экосистема Big Data !**

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

- **Фокус на решение задач в контексте бизнеса**
- **Фокус на решение задач в контексте науки о данных**
- **Уже сейчас : богатый функционал библиотек….**
- **и экосистема Big Data !**
- **Оптимизированные структуры данных**

и методы их обработки

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

- Фокус на решение задач в контексте бизнеса
- Фокус на решение задач в контексте науки о данных
- Уже сейчас : богатый функционал библиотек….
- и экосистема Big Data !
- Оптимизированные структуры данных

  и методы их обработки
- Мощность и богатство функционального программирования

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

- Фокус на решение задач в контексте бизнеса
- Фокус на решение задач в контексте науки о данных
- Уже сейчас : богатый функционал библиотек….
- и экосистема Big Data !
- Оптимизированные структуры данных

  и методы их обработки
- Мощность и богатство функционального программирования
- Но если очень хочется - можно всегда ООП

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH

Scala

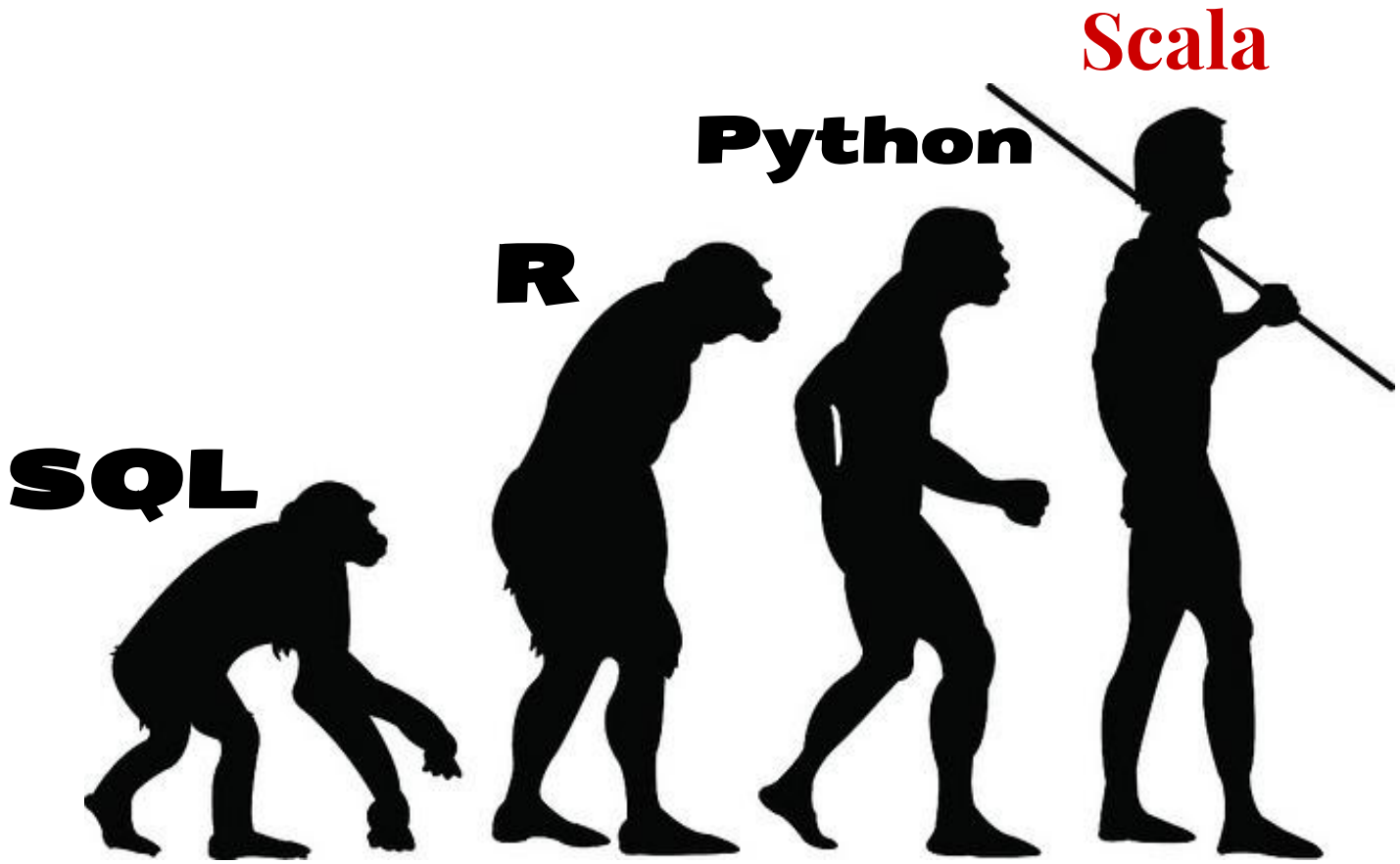
- Фокус на решение задач в контексте бизнеса
- Фокус на решение задач в контексте науки о данных
- Уже сейчас : богатый функционал библиотек….
- и экосистема Big Data !
- Оптимизированные структуры данных

  и методы их обработки
- Мощность и богатство функционального программирования
- Но если очень хочется - можно всегда ООП
- Типобезопасный код - наш выбор!

Спасибо за внимание!!

Вопросы!!

SCALABLE DATA-SCIENCE: GROWABLE & ENABLING GROWTH



@lievAnastazia