**1...................................Графы............................................**

1.1. ........................................Бизнес-кейсы................................

1.2. ........................................Теория......................................

**2...................................GRAPHX.........................................**

2.1.............................................Обзор инструмента.....................................

**3.................................SparkML/MLlib................................**

3.1.........................Возможности..................................................................

3.2. .....................Коллекции.........................................................................

3.3. .....................Linear Regression.................................................

3.3.1....................Linear Regression with SGD...............................

**4.........................Spark Optimization.........................................**

4.1.............

**Графы**

**Бизнес-кейсы**

http://jonathankinlay.com/2016/09/applications-graph-theory-finance/

http://www.turingfinance.com/computational-model-of-systemic-risk-for-the-banking-industry/

https://quantdare.com/graph-theory-finance/

http://blog.wolfram.com/2012/06/01/graph-theory-and-finance-in-mathematica/

https://neo4j.com/blog/5-noteworthy-use-cases-graph-technology-analytics/

https://blogs.oracle.com/bigdata/fraud-detection-use-cases-graph-technology

http://blog.know.bi/graph-databases-analytical-use-cases

https://spark.apache.org/docs/2.3.1/graphx-programming-guide.html

**Теория**

* Вершины/ Vertex ‑ представляют объекты из реальной жизни, люди. фильм.
* Edge ‑ представляют отношения из реальной жизни.
* Properties ‑ свойства ребер/
* Social networks ‑ Perhaps the most typical use case for a graph database are social networks, with their complex relationships and user activity.
* Fraud detection ‑ To uncover fraud in real-time, fast analysis of data relationships is essential, and graph databases provide the necessary performance.
* Knowledge graphs ‑ Used by search engines and businesses alike, knowledge graphs gather information from a wide variety of sources, allowing for better digital asset management and easier information retrieval.
* IT network analysis ‑ The management of networks and IT infrastructures revolves around complex interdependencies, and graph databases are inherently more suitable for this use case than relational databases.
* Recommendations ‑ Companies can use graph databases to power sophisticated recommendation engines to personalize products, content, and services.
* Identity management ‑ Graph databases enable efficient user activity tracking and quick authorizations and asset management.

The path to follow in understanding graphs is to think about data connections, the relationships among people and their data and naturally occurring networks like ecosystems, grids, or brain activity.

**GraphX**

**Обзор**

**SparkML/MLlib**

**Возможности**

**1. Feature extraction/transfomers/selectors:**

NLP обработка текстов, трансформация строковых данных в различные вектора (численные фичи)

**2. Basic statistics:**

Хи-квадрат, корреляции Пирсона и Спирмана, медиана, дисперсия.

**3.Regressions**

**4. Support Vector Machine**

**5. Naive Bayes classifier**

Спам фильтр как пример. Обучается с учителем.

**6. Decision Tree**

Предсказательный алгоритм.

**7.K-Means clustering**

**8. PCA/singular value decomposition**

Сокращение размерности

**9. Recommendations with ALS**

и многое другое....

Примеры кода использования библиотек:

1. mllib

https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples/mllib

2.ml

https://github.com/apache/spark/tree/master/examples/src/main/scala/org/apache/spark/examples/mllib

**SparkML/MLlib**

Коллекции

**Ссылки**

// https://data-flair.training/blogs/spark-mllib-data-types/

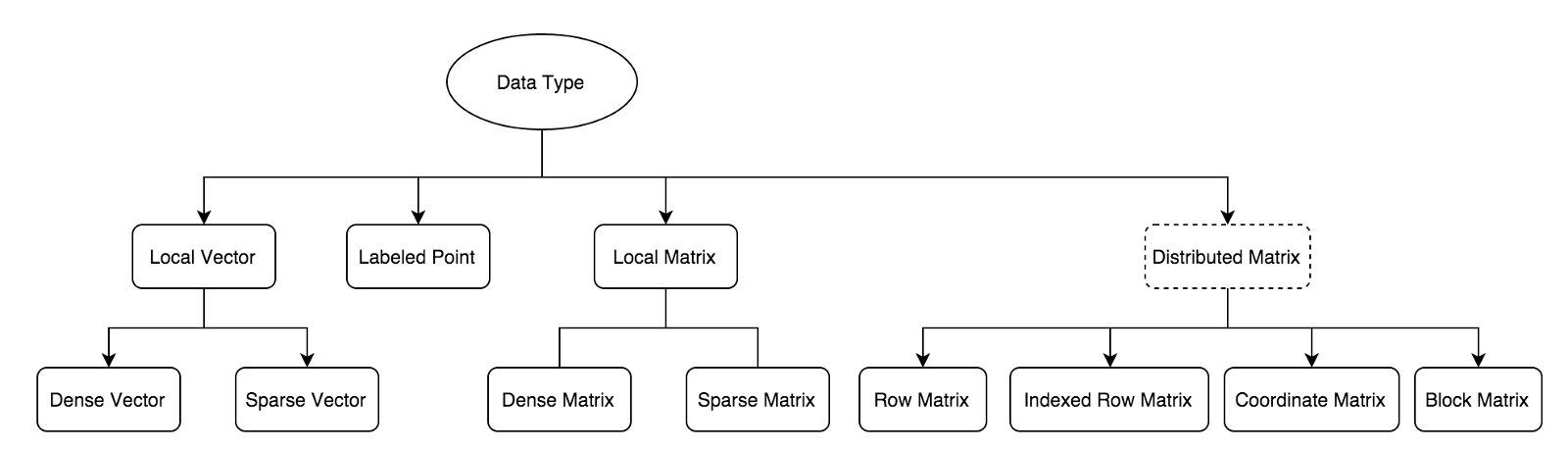
// https://medium.com/@rickynguyen/getting-started-with-spark-day-5-36b62a6d13bf

// http://www.cse.cuhk.edu.hk/~ericlo/teaching/bigdata/lab/5-MLlib/mllib-datatypes.html

// https://spark.apache.org/docs/2.2.0/mllib-data-types.html

**Основное**

Иерархия типов



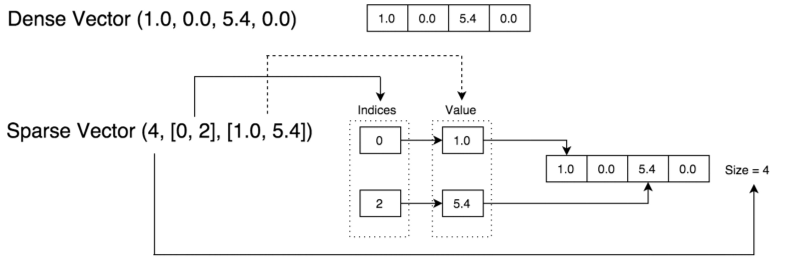
**Локальный вектор: Плотный и разряженный вектор**

* Локальный вектор - вектор лежащий на локальной машине, состоящий из целочисленных и двоичных значений.
* Включает в себя два подтипа: Плотный и разряженный вектора, второй отличается тем что имеет нулевые значения, также разраженный вектор состоит из нескольких подмассивов.

val dv: Vector = Vectors.dense(1.0, 0.0, 5.4, 0.0)

val sv1: Vector = Vectors.sparse(4, Array(0, 2), Array(1.0, 5.4))

val sv2**:** Vector = Vectors.sparse(4, Seq((0, 1.0), (2, 5.4)))



**LabelPoint (Отмеченные точки)**

* Отмеченные точки - это те же локальные вектора, но включающие в себя также результат/ответ - подходит для задач обучения с учителем.

For example:

val posP = LabelPoint(1.0, Vectors.dense(1.0, 0.0, 5.4, 0.0))

val negP = LabelPoint(0.0, Vectors.sparse(4, Array(0, 2), Array(1.0, 5.4)))

**Локальная матрица**

* Имеет целочисленные индексы строк и столбцов, включает в себя значения с плавающей точкой, хранится на локальном компьютере.
* Включает также в себя плотную и разряженную матрицу, хранится в сжатом разряженном колончатом формате.

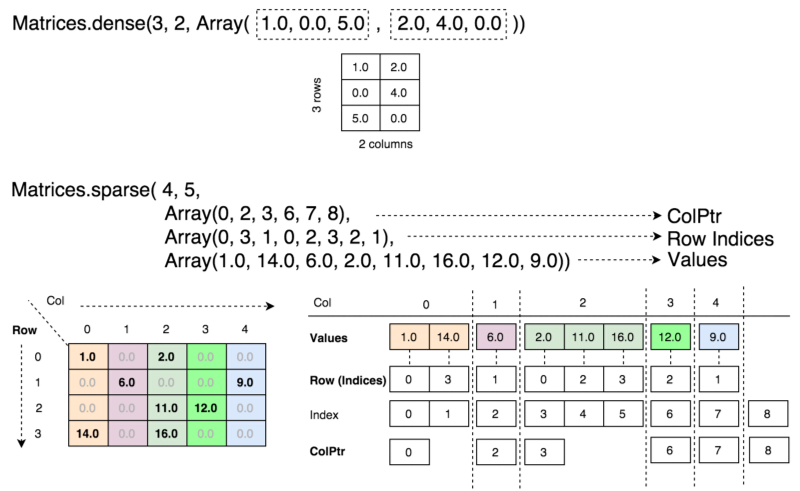
val dm: Matrix = Matrices.dense(3, 2, Array(1.0, 3.0, 5.0, 2.0, 4.0, 6.0))

val sm: Matrix = Matrices.sparse(4, 5,

Array(0, 2, 3, 6, 7, 8),

Array(0, 3, 1, 0, 2, 3, 2, 1),

Array(1.0, 14.0, 6.0, 2.0, 11.0, 16.0, 12.0, 9.0))



**Распределенная матрица**

* Содержит большие значения типа long для индексов строк и колонок. Значения являются значениями с плавающей точкой. Физически лежит распределенно на кластере в формате RDD.
* Включает в себя:

[RowMatrix](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.distributed.RowMatrix): RDD коллекции данных в виде векторов без индексов строк.

[IndexedRowMatrix](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.distributed.IndexedRowMatrix): RDD коллекции данных в виде векторов с индексами строк.

[CoordinateMatrix](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.distributed.CoordinateMatrix): определенная матрица вида [IndexedRow](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.distributed.IndexedRow)(row\_index, col\_index, value)

[BlockMatrix](http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.mllib.linalg.distributed.BlockMatrix): матрица включающая в себя матрицы вида (row\_index, col\_index, matrix)

**Raiting**

Тип данных необходимый для модели коллаборативной фильтрации.

Примеры всех типов данных для ML

*import* org.apache.spark.{SparkConf, SparkContext}

*import* org.apache.spark.mllib.linalg.distributed.\_

*import* org.apache.spark.mllib.linalg.{Matrices, Matrix, Vector, Vectors}

*import* org.apache.spark.mllib.regression.LabeledPoint

*import* org.apache.spark.rdd.RDD

*object* SimpleApp {

*def* main(args: Array[String]) {

*val* conf = *new* SparkConf().setAppName("Simple Application")

*val* sc = *new* SparkContext(conf)

*//Vectors*

*val* dv: Vector = Vectors.*dense*(1.0, 0.0, 3.0)

*val* sv1: Vector = Vectors.*sparse*(3, *Array*(0, 2), *Array*(1.0, 3.0))

*val* sv2: Vector = Vectors.*sparse*(3, Seq((0, 1.0), (2, 3.0)))

*//Label*

*val* pos = LabeledPoint(1.0, Vectors.*dense*(1.0, 0.0, 3.0))

*val* neg = LabeledPoint(0.0, Vectors.*sparse*(3, *Array*(0, 2), *Array*(1.0, 3.0)))

*//Local Matrix*

*val* dm: Matrix = Matrices.*dense*(3, 2, *Array*(1.0, 3.0, 5.0, 2.0, 4.0, 6.0))

*val* sm: Matrix = Matrices.*sparse*(4, 5,

*Array*(0, 2, 3, 6, 7, 8),

*Array*(0, 3, 1, 0, 2, 3, 2, 1),

*Array*(1.0, 14.0, 6.0, 2.0, 11.0, 16.0, 12.0, 9.0))

*//Row matrix*

*val* rows: RDD[Vector] = sc.parallelize(Seq(

Vectors.*dense*(1.0, 0.0, 5.4, 0.0),

Vectors.*dense*(1.0, 0.0, 5.4, 0.0),

Vectors.*dense*(1.0, 0.0, 5.4, 0.0)

))

*val* rowMatrix: RowMatrix = *new* RowMatrix(rows)

*//indexed row matrix*

*val* indexRows: RDD[IndexedRow] = sc.parallelize(Seq(

*IndexedRow*(0, Vectors.*dense*(1.0, 0.0, 5.4, 0.0)),

*IndexedRow*(2, Vectors.*dense*(1.0, 0.0, 5.4, 0.0))

))

*val* indexedRowMatrix: IndexedRowMatrix = *new* IndexedRowMatrix(indexRows)

*//Coordinate Matrix*

*val* coordinateMatrixEntry: RDD[MatrixEntry] = sc.parallelize(Seq(

*MatrixEntry*(0, 0, 1.2),

*MatrixEntry*(1, 0, 2.1),

*MatrixEntry*(6, 1, 3.7)

))

*val* coordinateMatrix: CoordinateMatrix = *new* CoordinateMatrix(coordinateMatrixEntry)

*//BlockMatrix*

*val* blocks = sc.parallelize(Seq(

((0, 0), Matrices.*dense*(3, 2, *Array*(1, 2, 3, 4, 5, 6))),

((1, 0), Matrices.*dense*(3, 2, *Array*(7, 8, 9, 10, 11, 12)))

))

*val* blockMatrix: BlockMatrix = *new* BlockMatrix(blocks, 3, 2)

}

}

**Регрессия**

**SGD**

Регрессия с использованием стохастического градиентного спуска.

Плюсы:

* Быстрая скорость работы

Недостатки:

* Нельзя использовать когда данные не подчинены нормальному распределению!

Аналоги: Линейная регрессия методом наименьших квадратов