### Objectives

Key objectives of this chapter:

- The Spark Machine Learning Library (MLlib)
- MLlib dense and sparse vectors and matrices
- Types of distributed matrices
- LIBSVM format
- Supported classification, regression and clustering algorithms

### 1.1 What is MLlib?

- Spark Machine Learning Library (MLlib) provides an array of high quality distributed Machine Learning (ML) algorithms

- The MLlib library implements a whole suite of statistical and machine learning algorithms (see Notes for details)

- MLlib provides tools for
  - Building processing workflows (e.g. feature extraction and data transformation),
  - Parameter optimization, and
  - ML model management for model saving and loading

- MLlib applications run on top of Spark and take full advantage of Spark's distributed in-memory design

- MLlib applications claim 10X+ faster performance for applications that implement similar algorithms created using Apache Mahout

  - Apache Mahout apps leverage Hadoop's MapReduce engine

### Notes:

MLlib 1.3 contains the following algorithms (source: https://spark.apache.org/mllib/):

- linear SVM and logistic regression
- classification and regression tree
random forest and gradient-boosted trees
recommendation via alternating least squares
clustering via k-means, Gaussian mixtures, and power iteration clustering
topic modeling via latent Dirichlet allocation
singular value decomposition
linear regression with L1- and L2-regularization
isotonic regression
multinomial naive Bayes
frequent itemset mining via FP-growth
basic statistics

1.2 Supported Languages

- Java
- Python
  - You will need to install the NumPy package as a dependency
- R
- Scala

- In our further discussion, we will be using Python to illustrate the main concepts and programming structures

- **Note:** Spark version 1.6 (the latest in the 1.* version series) requires Java 7+, Python 2.6+, R 3.1+, and Scala 2.10
1.3 MLlib Packages

- MLlib is divided into two packages:
  - spark.mllib
    - Contains the original Spark API built on top of RDDs
  - spark.ml
    - Contains higher-level API built on top of DataFrames
    - spark.ml is recommended if you use the DataFrames API which is more versatile and flexible
    - Facilitates ML processing pipeline construction

Notes:

Recently, the Spark MLlib team has started encouraging ML developers to contribute new algorithms to the spark.ml package and at the same time are saying, "Users should be comfortable using spark.mllib features and expect more features coming." [http://spark.apache.org/docs/1.6.0/mllib-guide.html]

1.4 Dense and Sparse Vectors

- MLlib supports local and distributed vectors and matrices
- Local vectors can be dense or sparse
  - A dense vector is a regular array of doubles
  - A sparse vector is backed by two parallel arrays: one for indices of elements that are present, and the other for double values for those elements
  - Values 1.0, 2.0, 0.0, 4.0 (a four element list) can be represented in dense format as
[1.0, 2.0, 0.0, 4.0]

✓ Same values in sparse format would be presented as

(4, [0,1,3], [1.0, 2.0, 4.0])

➢ The first element is the size of the list; the vector [0,1,3] holds the indices of present elements; the third element (at the index 2) is absent

1.5 Labeled Point

- A labeled point is an object that represents label/category with a local vector (dense or sparse) of its properties
- Used in supervised learning algorithms in MLlib
- A label is represented by a single double starting from 0.0 for the first category, 1.0 for the second, etc., which make it possible to use them in both regression and classification algorithms
  ◦ For binary classification cases, a label should be either 0.0 (for negative classification) or 1.0 (for positive classification)
- A label point is brought into code as an instance of the LabeledPoint class that carries the features (properties) and labels (categories) of the data asset
  ◦ Features of a point are packaged as a vector
1.6 Python Example of Using the LabeledPoint Class

```python
from pyspark.mllib.linalg import SparseVector
from pyspark.mllib.regression import LabeledPoint

# Features are represented by a dense feature vector:
dataPointA=LabeledPoint(0.0, [11.0, 2.2, 333.3, 4.444])
# The dataPointA is labeled as belonging to category 0.0
# Features are some measured properties of the data Point object
# (e.g. size, speed, duration, breath rate, etc.

# Features are represented by a sparse feature vector (elements at
# position 1 and 2 in the feature vector are zeroed out):
dataPointB=LabeledPoint(1.0, SparseVector(4, [0, 3], [11.0, 4.444]))
```

1.7 LIBSVM format

- **LIBSVM** is a compact text format for encoding data (usually representing training data sets)
- Widely used in MLlib to represent sparse feature vectors
- A file in **LIBSVM** format is shaped as a matrix in which each line is a space-delimited record that represents a labeled sparse feature (attribute / property) vector
- The layout is as follows:

  ```text
class_label index1:value1 index2:value2 ...
  ◦ where the numeric indices represent features; values are separated from indices via a colon (':')
  ◦ MLlib expects you to start class labeling from 0
  ◦ Feature indices are one-based in ascending order (1,2,3, etc.); where a feature is not present in the data record, it is omitted from the record
  ◦ **Note**: After loading in your MLlib application, the feature indices are
  ```
Notes:

LIBSVM a library (and its data format) for support vector machines [http://www.csie.ntu.edu.tw/~cjlin/libsvm/faq.html].

This resource [http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/] contains a great number of classification, regression, multi-label and string data sets stored in LIBSVM format.

1.8 An Example of a LIBSVM File

```
0 2:1 8:1 14:1 21:1 23:1 25:1
0 1:1 9:1 11:1 13:1 18:1 20:1
1 1:1 5:1 15:1 18:1 21:1 23:1
2 6:1 9:1 12:1 14:1 16:1 24:1
2 8:1 13:1 21:1 22:1 27:1 30:1
3 6:1 8:1 10:1 15:1 17:1 21:1
```

1.9 Loading LIBSVM Files

- MLlib provides the **MLUtils** class, which, among many other things, offers the method for loading a file in LIBSVM format:

```
from pyspark.mllib.util import MLUtils
dataSet=MLUtils.loadLibSVMFile(sc, "data/mllib/libsvm_data.dat")
```

- **Note:**
  - The resulting `dataSet` object is an RDD with the records stored as `LabeledPoint` objects
1.10 Local Matrices

- Matrices in MLlib are stored as vectors with their dimensions provided as parameters to help with the matrix layout (see next slide for an example).
- Like with vectors, matrices can be dense or sparse.
- Data in a matrix is stored in the Compressed Sparse Column (CSC) format.
  - More specifically, data is serialized into a vector column-wise.
- Example of CSC format:
  - If the original matrix data layout is
    
    \[
    \begin{pmatrix}
    1.0 & 2.0 & 3.0 \\
    4.0 & 5.0 & 6.0
    \end{pmatrix}
    \]
  - Then the matrix data is packed in the resulting CSC-encoded vector as follows:
    \[\begin{pmatrix}
    1.0, 1.4, 2.0, 5.0, 3.0, 6.0
    \end{pmatrix}\]

1.11 Example of Creating Matrices in MLlib

```scala
import org.apache.spark.mllib.linalg.{Matrix, Matrices}

# Create a dense matrix ((1.0, 2.0), (3.0, 4.0), (5.0, 6.0)) shaped
# in 3 rows, 2 columns; the first column will have values 1.0, 3.0,
# and 5.0; the second column will have values 2.0, 4.0, and 6.0
dm2 = Matrices.dense(3, 2, [1, 2, 3, 4, 5, 6])
```

Canada
821A Bloor Street West
Toronto, Ontario, M6G 1M1
1 866 206 4644
getinfo@webagesolutions.com

United States
744 Yorkway Place
Jenkintown, PA. 19046
1 877 517 6540
getinfousa@webagesolutions.com
Notes:

The Python syntax of the `dense()` and `sparse()` methods of the `Matrices` class is as follows:

```python
static dense(numRows, numCols, values)
Description: Create a DenseMatrix
```

```python
static sparse(numRows, numCols, colPtrs, rowIndices, values)
Description: Create a SparseMatrix
```

## 1.12 Distributed Matrices

- In MLlib, a distributed matrix is stored across a cluster of machines in one or more RDDs
- It uses long-typed (8-byte) row and column indices and double-typed values
- MLlib has implemented the following types of distributed matrices so far: `RowMatrix`, `IndexedRowMatrix`, `CoordinateMatrix`, and `BlockMatrix`
  - *Note:* You need to match your processing use case with the right distributed matrix type -- it is an advanced topic not reviewed here, please refer to the Spark original documentation: [http://spark.apache.org/docs/1.6.0/mllib-data-types.html#distributed-matrix](http://spark.apache.org/docs/1.6.0/mllib-data-types.html#distributed-matrix)
- The main idea behind the distributed matrices is based on splitting the underlying matrix into a set of vectors and distribute the processing task across the cluster of machines using the `parallelize()` method of the Spark Context object
1.13 Example of Using a Distributed Matrix

- The following example shows how to use the `RowMatrix` type
- In the `RowMatrix` distributed matrix type, each row in an RDD is a local vector

```python
from pyspark.mllib.linalg.distributed import RowMatrix

# Create an RDD of vectors
t1 = [1.0, 2.0, 3.0]
t2 = [4.0, 5.0, 6.0]

t = sc.parallelize([t1, t2])

distributedMatrix = RowMatrix(t)
```

1.14 Classification and Regression Algorithm

- According to Spark documentation (http://spark.apache.org/docs/1.6.0/mllib-classification-regression.html), MLlib supports the following algorithms for classification and regression in its `spark.mllib` package:

<table>
<thead>
<tr>
<th>Problem Type</th>
<th>Supported Methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Classification</td>
<td>linear SVMs, logistic regression, decision trees, random forests, gradient-boosted trees, naive Bayes</td>
</tr>
<tr>
<td>Multiclass Classification</td>
<td>logistic regression, decision trees, random forests, naive Bayes</td>
</tr>
<tr>
<td>Regression</td>
<td>linear least squares, Lasso, ridge regression, decision trees, random forests, gradient-boosted trees, isotonic regression</td>
</tr>
</tbody>
</table>
1.15 Clustering

- According to Spark documentation (http://spark.apache.org/docs/1.6.0/mllib-clustering.html), MLlib supports the following algorithms for clustering in its spark.mllib package:
  - K-means
  - Gaussian mixture
  - Power iteration clustering (PIC)
  - Latent Dirichlet allocation (LDA)
  - Bisecting k-means
  - Streaming k-means

1.16 Summary

- In this chapter we have reviewed the following topics:
  - MLlib packages and supported languages
  - The ways to create dense and sparse vectors and matrices
  - Types of distributed matrices
  - LIBSVM format
  - Supported classification, regression and clustering algorithms