UNIVERSITÄT BONN

SMART DATA ANALYTICS
FROM DATA TO KNOWLEDGE

MA-INF 4223 - Lab Distributed Big Data Analytics

MLlib and BigDL

Dr. Hajira Jabeen, Gezim Sejdiu, Denis Lukovnikov

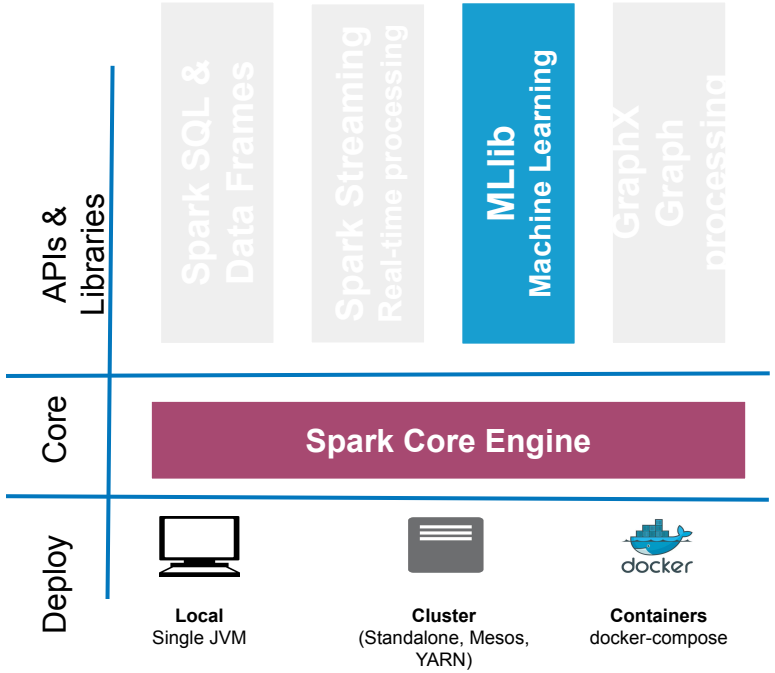
Summer Semester 2019

Lesson objectives

1. After completing this lesson, you should be able to:
   a. Understand and use
      i. Spark MLlib
      ii. BigDL

Spark ML

Spark ML

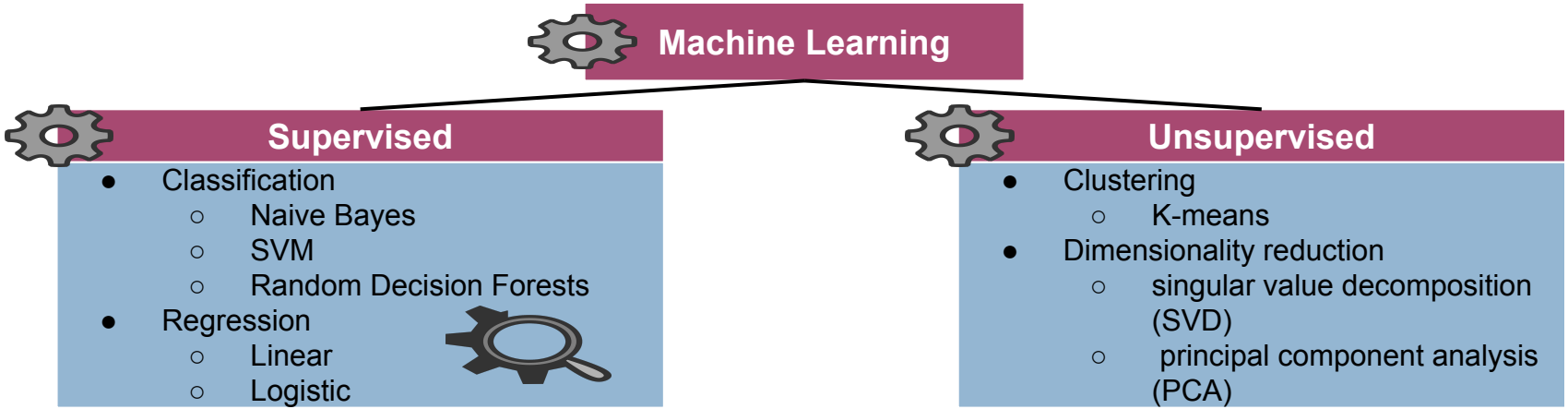

APIs & Libraries
Spark SQL & Data Frames
Spark Streaming Real-time processing
MLlib Machine Learning
GraphX Graph processing
Core
Spark Core Engine
Deploy
docker
Local
Single JVM
Cluster
(Standalone, Mesos, YARN)
Containers
docker-compose

Spark ML

1. MLlib is a standard component of Spark providing machine learning primitives on top of Spark
2. It provides scalable machine learning, statistics, math algorithms
3. Supports out-of-the-box most popular machine learning algorithms like Linear regression, Logistic regression, Decision Trees
4. Is available in Scala, Java, Python, and R

ML Algorithms overview

1. Machine learning are separated in two major types of algorithms :
   a. Supervised - labeled data in which both, input and output are provided to the algorithm
   b. Unsupervised - do not have the outputs in advance



Spark ML-pipelines

1. Uniform set of APIs for creating and tuning data processing/machine learning pipelines
2. Core concepts:
   a. DataFrame: RDD with named columns. SQL-like syntax and other core RDD operations
   b. Transformer: DataFrame ⇒ DataFrame. Eg., features to predictions (classifier)
   c. Estimator: DataFrame ⇒ Transformer. e.g., learning algorithm
   d. Pipeline: Chain of Transformers and Estimators. Specifies the data flow

Spark ML - pipelines: Transformer

- A Transformer transforms one DataFrame to another
- It implements a method `transform()`
- Both feature extractors and (un)trained models are Transformers, because they augment input data with features resp. predictions.

Spark ML - pipelines: Estimator

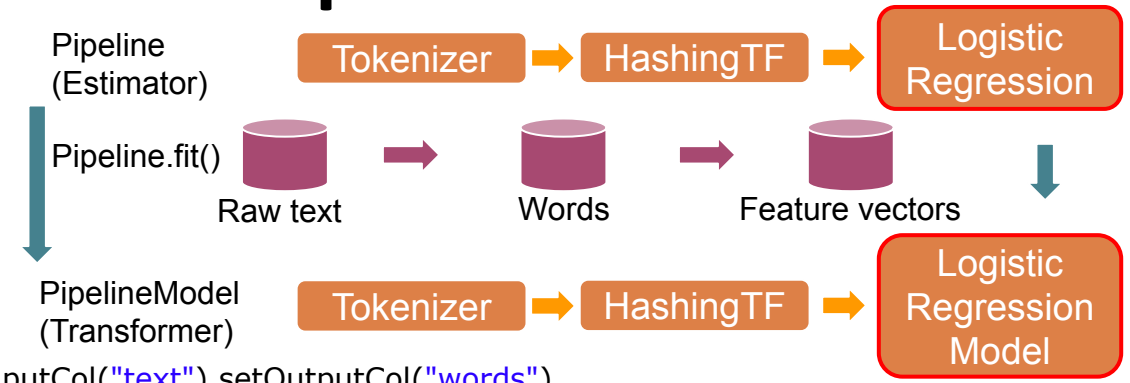
- An Estimator abstraction uses an algorithm which fits a model into a DataFrame
  - Learning algorithms are Estimators
  - Estimators produce models (models are Transformers)
- It implements a method `fit()`

Spark ML - pipelines: Pipeline

- Pipeline: is an Estimator
  - produces a Pipeline-model (a Transformer)
- consists of Transformers and/or Estimators.
- When .fit() is called on the Pipeline:
  - The stages are run in specified order
  - If stage is Transformer, .transform() is called
  - If stage is Estimator, .fit() is called
- pipeline.fit() produces a PipelineModel, where all Estimators are replaced by the Transformers they produced
- ⇒ can easily specify multiple tasks to be trained in a single pipeline and used at test time

Spark ML-pipelines Example

1. Split text into words => convert numerical features => generate a prediction model




```
val tokenizer = new Tokenizer().setInputCol("text").setOutputCol("words")
val hashingTF = new HashingTF().setNumFeatures(1000).setInputCol(tokenizer.getOutputCol)
.setOutputCol("features")
val lr = new LogisticRegression().setMaxIter(10).setRegParam(0.01)
val pipeline = new Pipeline().setStages(Array(tokenizer, hashingTF, lr))
val model = pipeline.fit(training.toDF)
val test = sc.parallelize(Seq(
Document(4L, "spark i j k"),
Document(5L, "l m n"),
Document(6L, "mapreduce spark"),
Document(7L, "apache hadoop")))
val predictions = model.transform(test.toDF)
```

References

[1]. MLlib: Machine Learning in Apache Spark by Meng, Xiangrui, Joseph K. Bradley, Burak Yavuz, Evan R. Sparks, Shivaram Venkataraman, Davies Liu, Jeremy Freeman, D. B. Tsai, Manish Amde, Sean Owen, Doris Xin, Reynold Xin, Michael J. Franklin, Reza Bosagh Zadeh, Matei Zaharia and Ameet Talwalkar *in Journal of Machine Learning Research 17*, 2016.

[2]. “Machine Learning Library (MLlib) Guide” - http://spark.apache.org/docs/latest/ml-guide.html

BigDL

BigDL Intro

1. Deep Learning in Spark
2. ⇒ write deep learning applications as Spark programs
3. Why (advantages)?
   a. big data lives on cluster → would be nice to perform learning/inference on cluster itself (otherwise need to copy)
4. Disadvantage:
   a. no GPU acceleration
5. can run existing models from popular DL frameworks in BigDL:
   a. TensorFlow, Caffe, Keras, Torch
6. compatible with Spark ML Pipelines
7. whitepaper: https://bigdl-project.github.io/0.8.0/#whitepaper/

Deep learning intermezzo

Check this out for more details :
https://github.com/AskNowQA/QA-Tutorial/blob/tutorial/slides/Session%203%20-%20Deep%20Learning.pdf

How SGD normally works (single node)

a. data: list of *(x, y)* pairs
b. model with parameters θ: *m*
c. during training, parameters θ are repeatedly updated such that the loss $L(m(x;\theta), y)$ between model predictions $m(x;\theta)$ and real label *y* is minimized
d. the data are split into mini-batches and fed to the model

How SGD normally works (single node)

typical training loop:

```
while not done:
    # (loops as long as the model predictions are not good enough) ← iteration goes over all data
    for batch in batches:
        # (iterates over all available data) ← one loop over all data is one epoch
        compute gradient of loss w.r.t. params θ
        # (gradient points in the direction that locally increases loss)
        update params θ by taking small step against the gradient
```

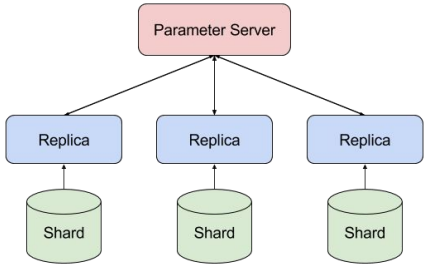
Distributing SGD over multiple nodes

1. split data in N pieces and distribute over N nodes
2. make N copies of model and broadcast over the N nodes
3. at every update, each node:
   a. draws a local mini-batch from its local data
   b. computes loss and local gradient based on local mini-batch
   c. (no inter-node communication so far, same as normal SGD)
4. but: each node will have a different gradient ⇒ using only local gradient for updating local model will result at different model param values at every node

Distributing SGD over multiple nodes

1. ⇒ local gradients must be synchronized across nodes and the "master" model updated and distributed before nodes compute updates for next local mini-batches

2. one solution: central parameter server (one machine holds "master" copy of params)
   a. easy to implement
   b. easier for async SGD
   c. slower
   d. less scalable



img from: https://seba-1511.github.io/dist_blog/

Parameter synchronization

BigDL: distributed AllReduce on Spark:

1. divide params from all local model copies into M identically partitioned groups (partition parameters)
2. send the local gradients for each group to same node (gather by group)
3. sum all gathered local gradients and update Mth param group (update group)
4. broadcast each (updated) param group to all nodes for next mini-batch (redistribute)

BigDL SGD

data-parallel training using synchronous mini-batch SGD

(SGD = stochastic gradient descent)

BigDL SGD

data-parallel training using synchronous mini-batch SGD

1. data-parallel: data is distributed over different cluster nodes
   a. helpful when a lot of data but model is relatively small (fits comfortably on single node)
   b. different nodes compute an update for all model params for their own part of the mini-batch
2. (vs model-parallel: different, disjoint groups of model parameters on different nodes)

BigDL SGD

data-parallel training using **synchronous** mini-batch SGD

- synchronous SGD
  a. gradients computed by different nodes (using different sub-mini-batches) are merged after all nodes are done computing
  b. need to wait until all nodes are done computing
- (vs asynchronous SGD: once a node is done, its param update is applied to “master” param copy and used everywhere before other nodes are done computing)
  a. don’t need to wait for all nodes to finish
  b. BUT: stale gradients: a longer-running job’s gradients were computed using older param values but applied to newer param values

BigDL: practical

1. Data transformation
2. Model definition
3. Training
4. Inference

BigDL data transformation

- Before data can be used for training, they must be transformed to either:
  - RDD[bigdl.dataset.Sample] or
  - bigdl.dataset.DataSet
- Python example:

```
<RDD>.map(lambda (features, label): Sample.from_ndarray(features, label))
```

- Scala example:

```
<RDD>.map(x => Sample(Tensor(x._1), x._2))
```

- BigDL also provides various data transformers (see API)

BigDL model definition

- This is where we declare our neural network architecture
- BigDL provides various layers and Sequential and Functional APIs to arrange layers into Models

BigDL model definition: *Sequential API*

Python:

```
model = Sequential()
model.add(Linear(...))
...
model.add(Sigmoid())
```

Scala:

```
val model = Sequential()
model.add(Linear(...))
...
model.add(Sigmoid())
```

Sequential API also supports branching and merging layers for more complex models (see API).

Although *Functional API* might be easier to use for complex models.

BigDL model definition: *Functional API*

Python:

```
linear1 = Linear(...)()
relu1 = ReLU()(linear1)
linear2 = Linear(...)(relu1)
relu2 = ReLU()(linear2)
m = Model([linear1], [relu1, relu2])
```

Scala:

```
val linear1 = Linear(...).inputs()
val relu1 = ReLU().inputs(linear1)
val linear2 = Linear(...).inputs(relu1)
val relu2 = ReLU().inputs(linear2)
val m = Graph(Seq[linear1], Seq[relu1, relu2])
```

(⇒ Defines model with two outputs)

⟶ allows for arbitrary branching by explicitly specifying inputs to each layer

BigDL: training

- Training: change model parameters to minimize loss on given data
- Neural networks: stochastic gradient descent (SGD)
  - Take batch
  - Feed through network and compute loss (forward)
  - Compute gradient (backward) -- direction in parameter space in which to move to increase loss
- BigDL provides Optimizer:
  - Implements training algorithm (forward and backward)
  - As Spark jobs

BigDL: training

- Optimizer needs:
  1. Model
  2. Data
  3. Loss function
- Other Optimizer settings (see API/Guide):
  1. Batch size
  2. Termination criterion (e.g. number of epochs)
  3. Optimization algorithm (SGD/Adam/...)
  4. Validation
  5. Monitoring

BigDL: training

- Optimizer example (Python):

```
optimizer = Optimizer(
    model=model,
    training_rdd=train_data,
    criterion=MSECriterion(),
    optim_method=SGD(learningrate=0.1, learningrate_decay=0.0001),
    end_trigger=MaxEpoch(100),
    batch_size=4)
trained_model = optimizer.optimize()
```

- Scala API provides same functionality but slightly different API

BigDL: training + validation

- During training, we want to see how loss evolves on a unseen data (validation set)
- BigDL:

```
optimizer.set_validation(
    batch_size=batch_size,
    val_rdd=validation_rdd,
    trigger=EveryEpoch(),
    val_method=[Loss(<SOMELOSS>), <SOMEMETRIC>]
)
```

BigDL: prediction and evaluation

- Prediction (Python) :

  ```
  trained_model.predict(new_data)
  ```

- Evaluation (Python):

  ```
  model.evaluate(test_rdd, batch_size, [<LIST_OF_METRICS>])
  ```

BigDL: Monitoring

- monitor losses and metrics on training and validation sets
- BigDL provides summary writers that record training loss and validation metrics:

```
train_summary = TrainSummary(log_dir="./trainlogs/", app_name=<APPNAME>)
valid_summary = ValidationSummary(log_dir="./trainlogs/",
app_name=<APPNAME>)
optimizer.set_train_summary(train_summary)
optimizer.set_val_summary(valid_summary)
```

- The summaries can be viewed in TensorBoard
- Summaries can also be inspected and plotted:

```
np.array(train_summary.read_scalar("Loss"))
```

BigDL: Spark ML integration

- DLEstimator is specified by the model and criterion (and input and output sizes)
- DLEstimator is fitted to a DataFrame and produces a DLModel

- Parallel interface with DLClassifier and DLClassifierModel (see API/docs)

⇒ similar functionality to Optimizer but allows integration into Spark ML Pipelines

Hands-On

- Download Spark 2.2, unpack somewhere (/opt/spark…)
- Set SPARK_HOME var to unpacked Spark
- Download BigDL 0.7, unpack anywhere
- Set BIGDL_HOME var to unpacked BigDL directory
- do `pip install bigdl==0.7` somewhere
- download https://gist.github.com/lukovnikov/461d1165ea04317d2be6b66995ffa73c
- start jupyter by running the script

At Home:

1. Reading:
   a. Read “Pattern Recognition and Machine Learning” by Bishop
   b. Read “Deep Learning” by Courville et al.
   c. *(or check some blog posts/tutorials)*
   d. Check out the MLlib programming guide
   e. Read the BigDL whitepaper
   f. Check out the BigDL programming guide
   g. Check out the BigDL tutorials (https://github.com/intel-analytics/BigDL-Tutorials/ ← Python)
2. Complete the notebooks
3. Convert the mnist_cnn notebook to use MLlib’s Pipeline API
4. Check out PyTorch!

UNIVERSITÄT BONN

SMART DATA ANALYTICS
FROM DATA TO KNOWLEDGE

THANK YOU !

< http://sda.cs.uni-bonn.de/teaching/dbda/ >

Dr. Hajira Jabeen
jabeen@cs.uni-bonn.de
Room 1.062
(Appointment per e-mail)

Gezim Sejdiu
sejdiu@cs.uni-bonn.de
Room 1.068
(Appointment per e-mail)

Denis Lukovnikov
lukovnikov@cs.uni-bonn.de
Room 1.064
(Appointment per e-mail)