MACHINE LEARNING ON SPARK

Yiheng Wang (yiheng.wang@intel.com)

Big Data Technologies, Software and Service Group, Intel

Introduction

Intel Big Data Technology team

- Active open source development
- Spark, Hadoop, HBase, Hive, Sentry, Storm, etc
- ~30 project committers in the team

My focusing area

- Large scale machine learning, deep learning
- Next generations of Big Data analytics solutions with Intel customers

Apache Spark MLlib

Make practical machine learning scalable and easy.

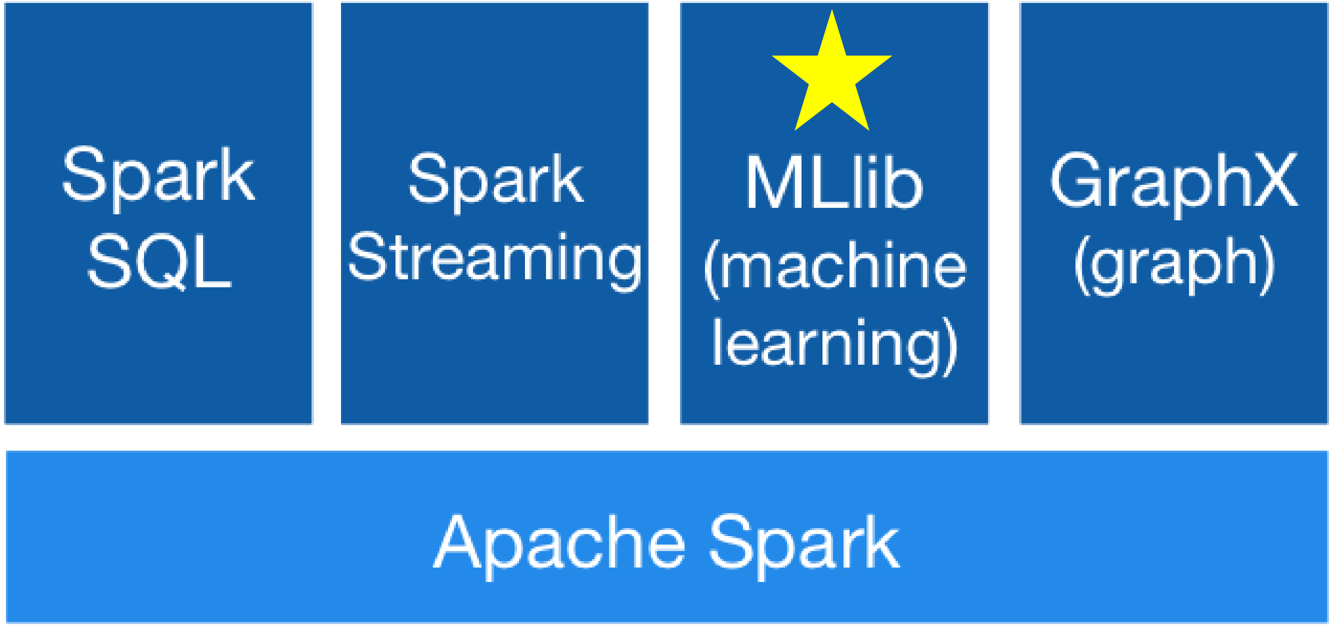
Outline

- Overview
- Machine Learning Pipeline
- Feature Engineering
- ML Algorithms
- Tuning

OVERVIEW

An overview of Apache Spark MLlib

One Platform to Rule Them All



Build an End-2-end Solution

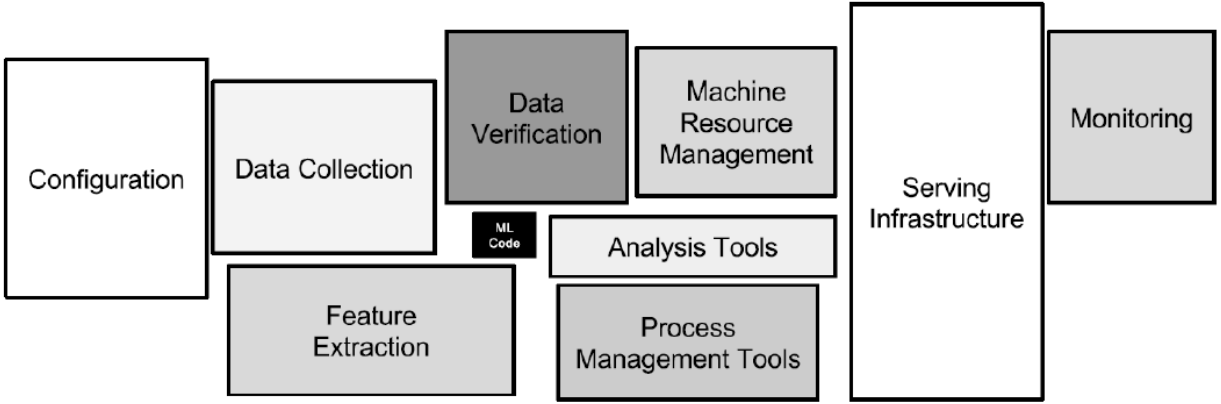


Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small black box in the middle. The required surrounding infrastructure is vast and complex.

"Hidden Technical Debt in Machine Learning Systems",
Google, NIPS 2015 Paper

Build an End-2-end Solution

Challenges

- compatible with different data source format
- performance and scalability
- stability & fault tolerant
- data statistic analyze

Spark and other component

- feature engineering
- different machine learning algorithms
- hyper-parameter

MLlib

What’s in MLlib

MLlib provides

- ML Algorithms
  - Classification, regression, clustering and collaborative filtering
- Featurization
  - feature extraction, transformation, dimensionality reduction, and selection
- Pipelines
- Persistence
- Utilities
  - linear algebra, statistics, model tunning, etc

ML and MLlib

Wait, there're two libraries under MLlib

- **MLlib RDD-based API**
- **MLlib DataFrame-based API**

DataFrame-based API is primary API

Language

Java

Scala

python

R
Data Mining and
Machine Learning

MACHINE LEARNING PIPELINE

People can have the Model T in any color - so long as it's black. - Henry Ford

Machine Learning Pipelines

We will take a look at machine learning pipeline in this order

- DataFrame
- Transformer and Estimator
- A Simple Example

Sandbox enviroment

https://github.com/yiheng/OReillyAIConf#sandbox-environment

DataFrame

DataFrame is a table

- scalable
- schema
- named columns
- can contain vectors, text, images, and structured data
- just like the one in pandas

DataFrame

```
# Defines a Python list storing one JSON object.
json_strings = ['{"name":"Han Meimei","address":{"city":"Beijing", "province":"Beijing"}}',
            '{"name":"Li Lei","address":{"city":"Hangzhou", "province":"Zhejiang"}}']

# Defines an RDD from the Python list.
peopleRDD = sc.parallelize(json_strings)

# Creates an DataFrame from an RDD[String].
people = spark.read.json(peopleRDD)

people.show()

people.printSchema()
```

DataFrame

```
+--------------------+-----------+
|             address|       name|
+--------------------+-----------+
|   [Beijing,Beijing]|Han Meimei|
|[Hangzhou,Zhejiang]|     Li Lei|
+--------------------+-----------+
```

```
root
 |-- address: struct (nullable = true)
 |    |-- city: string (nullable = true)
 |    |-- province: string (nullable = true)
 |-- name: string (nullable = true)
```

Transformer and Estimator

Estimator

Transformer

Transformer

Convert one DataFrame into another

- feature transformers
- learned models
- transform() method
- appending one or more columns

Transformer

```
from pyspark.ml.feature import Tokenizer

sentenceDataFrame = spark.createDataFrame([
    (0, "Hi I heard about Spark"),
    (1, "I wish Java could use case classes"),
    (2, "Logistic,regression,models,are,neat")
], ["id", "sentence"])

tokenizer = Tokenizer(inputCol="sentence", outputCol="words")
tokenized = tokenizer.transform(sentenceDataFrame)
tokenized.select("sentence", "words").show(truncate=False)
```

Transformer

sentence	words
Hi I heard about Spark	[hi, i, heard, about, spark]
I wish Java could use case classes	[i, wish, java, could, use, case, classes]
Logistic,regression,models,are,neat	[logistic,regression,models,are,neat]

Estimator

Abstracts the concept of a learning algorithm or any algorithm that fits or trains on data

- fit(), which accept a DataFrame and produce a tranformer

Estimator

```
from pyspark.ml.feature import Word2Vec

# Input data: Each row is a bag of words from a sentence or document.
documentDF = spark.createDataFrame([
    ("Hi I heard about Spark".split(" "), ),
    ("I wish Java could use case classes".split(" "), ),
    ("Logistic regression models are neat".split(" "), )
], ["text"])

# Learn a mapping from words to Vectors.
word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="text", outputCol="result")
model = word2Vec.fit(documentDF)

result = model.transform(documentDF)
result.show(truncate=False)
```

Estimator

text	result
[Hi, I, heard, about, Spark]	[0.007542145531624556,-0.037311234138906,0.017764256894588472]
[I, wish, Java, could, use, case, classes]	[-0.01725128452692713,0.030733417087633694,0.04699897639719503]
[Logistic, regression, models, are, neat]	[0.1010503351688385,-0.04308200553059578,0.005826892331242561]

Pipeline - A Simple Example

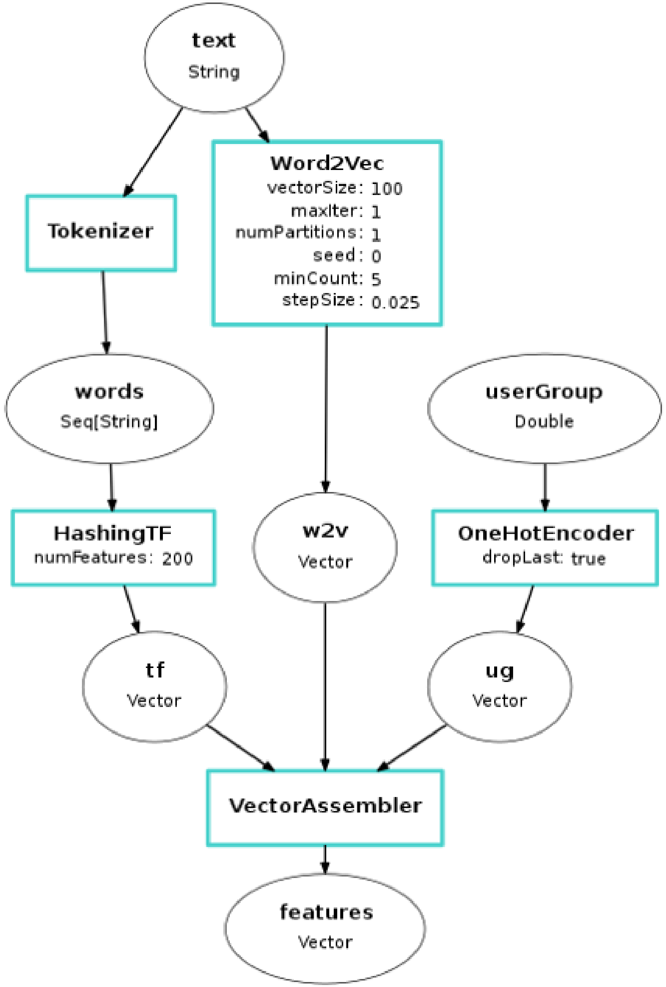
Data	Label
When I bought this lamp, I had no idea what I was getting into. It’s amazing the kind of low quality you find online.	3
Dude the laptops really cool and I gotta say its much better than the other one I got	2
I had to get a gift for my dad, and I saw this kite, It reminded me of when I was kid	5

Pipeline - A Simple Example

How to convert sentence to trainable vector

- tokenizer
- embedding
- encoder label
- …

https://databricks.com/blog/2015/01/07/ml-pipelines-a-new-high-level-api-for-mllib.html



Pipeline - A Simple Example

```
from pyspark.ml.feature import *
from pyspark.ml import Pipeline
tok = Tokenizer(inputCol="text", outputCol="words")
htf = HashingTF(inputCol="words", outputCol="tf", numFeatures=200)
w2v = Word2Vec(inputCol="text", outputCol="w2v")
ohe = OneHotEncoder(inputCol="userGroup", outputCol="ug")
va = VectorAssembler(inputCols=["tf", "w2v", "ug"], outputCol="features")
pipeline = Pipeline(stages=[tok,htf,w2v,ohe,va])
```

FEATURE ENGINEERING

Sometimes good feature engineering is better than more powerful model

Feature Engineering

MLlib provide rich feature engineering algorithms, roughly divided into

- Extraction
  - Extracting features from “raw” data
- Transformation
  - Scaling, converting, or modifying features
- Selection
  - Selecting a subset from a larger set of features
- Locality Sensitive Hashing (LSH)

Overview

Feature Extractors

- TF-IDF
- Word2Vec
- CountVectorizer

Feature Transformers

- Tokenizer
- StopWordsRemover
- nn-gram
- Binarizer
- PCA
- PolynomialExpansion
- Discrete Cosine Transform (DCT)
- StringIndexer
- IndexToString
- OneHotEncoder
- VectorIndexer
- Interaction
- Normalizer
- StandardScaler
- MinMaxScaler

Feature Transformers

- MaxAbsScaler
- Bucketizer
- ElementwiseProduct
- SQLTransformer
- VectorAssembler
- QuantileDiscretizer
- Feature Selectors
- VectorSlicer
- RFormula
- ChiSqSelector

Locality Sensitive Hashing

- LSH Operations
  - Feature Transformation
  - Approximate Similarity Join
  - Approximate Nearest Neighbor Search
- LSH Algorithms
  - Bucketed Random Projection for Euclidean Distance
  - MinHash for Jaccard Distance

VectorAssembler

```
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler

dataset = spark.createDataFrame(
    [(0, 18, 1.0, Vectors.dense([0.0, 10.0, 0.5]), 1.0)],
    ["id", "hour", "mobile", "userFeatures", "clicked"])

assembler = VectorAssembler(
    inputCols=["hour", "mobile", "userFeatures"],
    outputCol="features")

output = assembler.transform(dataset)
print("Assembled columns 'hour', 'mobile', 'userFeatures' to vector column 'features'")
output.show(truncate=False)
```

VectorAssembler

```
Assembled columns 'hour', 'mobile', 'userFeatures' to vector column 'features'
+---+----+------+--------------+-------+-----------------------+
|id |hour|mobile|userFeatures  |clicked|features               |
+---+----+------+--------------+-------+-----------------------+
|0  |18  |1.0   |[0.0,10.0,0.5]|1.0    |[18.0,1.0,0.0,10.0,0.5]|
+---+----+------+--------------+-------+-----------------------+
```

QuantileDiscretizer

```
from pyspark.ml.feature import QuantileDiscretizer

data = [(0, 18.0), (1, 19.0), (2, 8.0), (3, 5.0), (4, 2.2)]
df = spark.createDataFrame(data, ["id", "hour"])

discretizer = QuantileDiscretizer(numBuckets=3, inputCol="hour", outputCol="result")

result = discretizer.fit(df).transform(df)
result.show()
```

QuantileDiscretizer

id	hour	result
0	18.0	2.0
1	19.0	2.0
2	8.0	1.0
3	5.0	1.0
4	2.2	0.0

ChiSqSelector

```
from pyspark.ml.feature import ChiSqSelector
from pyspark.ml.linalg import Vectors

df = spark.createDataFrame([
    (7, Vectors.dense([0.0, 0.0, 0.5, 1.0]), 1.0,),
    (8, Vectors.dense([0.0, 1.0, 0.0, 0.0]), 0.0,),
    (9, Vectors.dense([1.0, 0.0, 0.5, 0.1]), 0.0,)], ["id", "features", "clicked"])

selector = ChiSqSelector(numTopFeatures=1, featuresCol="features",
                         outputCol="selectedFeatures", labelCol="clicked")

result = selector.fit(df).transform(df)

print("ChiSqSelector output with top %d features selected" %
selector.getNumTopFeatures())
result.show()
```

ChiSqSelector

```
ChiSqSelector output with top 1 features selected
+---+------------------+-------+----------------+
| id|          features|clicked|selectedFeatures|
+---+------------------+-------+----------------+
|  7|[0.0,0.0,0.5,1.0]|    1.0|           [1.0]|
|  8|[0.0,1.0,0.0,0.0]|    0.0|           [0.0]|
|  9|[1.0,0.0,0.5,0.1]|    0.0|           [0.1]|
+---+------------------+-------+----------------+
```

ML ALGORITHMS

The real problem is not whether machines think, but whether men do. - B.F. Skinner

ML Algorithms

MLlib provides many machine learning algorithms, they can be roughly divided into

- classification and regression
- clustering
- collaborative filtering

Overview

Classification and regression

- Logistic regression(Binomial / Multinomial)
- Decision Tree
- Random Forest
- Gradient-boosted tree
- Multilayer perceptron classifier
- One-vs-All
- Naïve Bayes
- Linear regression
- Generalized linear regression
- Survival regression
- Isotonic regression

Classification and regression

- K-means
- Latent Dirichlet allocation (LDA)
- Bisecting k-means
- Gaussian Mixture Model (GMM)

Collaborative filtering

- Collaborative filtering

Naive Bayes code part1

```
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
import urllib2

html = urllib2.urlopen('https://raw.githubusercontent.com/apache/spark/branch-
2.1/data/mllib/sample_libsvm_data.txt') .read()
text_file = open("sample_libsvm_data.txt", "w")
text_file.write(html)
text_file.close()

# Load training data
data = spark.read.format("libsvm").load("sample_libsvm_data.txt")

# Split the data into train and test
splits = data.randomSplit([0.6, 0.4], 1234)
train = splits[0]
test = splits[1]
```

Naive Bayes code part1

```
# create the trainer and set its parameters
nb = NaiveBayes(smoothing=1.0, modelType="multinomial")

# train the model
model = nb.fit(train)

# select example rows to display.
predictions = model.transform(test)
predictions.show()

# compute accuracy on the test set
evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction",
                                              metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print("Test set accuracy = " + str(accuracy))
```

Naive Bayes

label	features	rawPrediction	probability	prediction
0.0	(692,[95,96,97,12...	[-174115.98587057...	[1.0,0.0]	0.0
0.0	(692,[98,99,100,1...	[-178402.52307196...	[1.0,0.0]	0.0
0.0	(692,[100,101,102...	[-100905.88974016...	[1.0,0.0]	0.0
0.0	(692,[123,124,125...	[-244784.29791241...	[1.0,0.0]	0.0
0.0	(692,[123,124,125...	[-196900.88506109...	[1.0,0.0]	0.0
0.0	(692,[124,125,126...	[-238164.45338794...	[1.0,0.0]	0.0
0.0	(692,[124,125,126...	[-184206.87833381...	[1.0,0.0]	0.0
0.0	(692,[127,128,129...	[-214174.52863813...	[1.0,0.0]	0.0
0.0	(692,[127,128,129...	[-182844.62193963...	[1.0,0.0]	0.0
0.0	(692,[128,129,130...	[-246557.10990301...	[1.0,0.0]	0.0
0.0	(692,[152,153,154...	[-208282.08496711...	[1.0,0.0]	0.0
0.0	(692,[152,153,154...	[-243457.69885665...	[1.0,0.0]	0.0
0.0	(692,[153,154,155...	[-260933.50931276...	[1.0,0.0]	0.0
0.0	(692,[154,155,156...	[-220274.72552901...	[1.0,0.0]	0.0
0.0	(692,[181,182,183...	[-154830.07125175...	[1.0,0.0]	0.0
1.0	(692,[99,100,101,...	[-145978.24563975...	[0.0,1.0]	1.0
1.0	(692,[100,101,102...	[-147916.32657832...	[0.0,1.0]	1.0
1.0	(692,[123,124,125...	[-139663.27471685...	[0.0,1.0]	1.0
1.0	(692,[124,125,126...	[-129013.44238751...	[0.0,1.0]	1.0
1.0	(692,[125,126,127...	[-81829.799906049...	[0.0,1.0]	1.0

only showing top 20 rows

Test set accuracy = 1.0

Kmeans code part1

```
from pyspark.ml.clustering import KMeans
import urllib2

html = urllib2.urlopen('https://raw.githubusercontent.com/apache/spark/branch-
2.1/data/mllib/sample_kmeans_data.txt') .read()
text_file = open("sample_kmeans_data.txt", "w")
text_file.write(html)
text_file.close()

# Loads data.
dataset = spark.read.format("libsvm").load("sample_kmeans_data.txt")
dataset.show(truncate=False)
```

Kmeans code part2

```
# Trains a k-means model.
kmeans = KMeans().setK(2).setSeed(1)
model = kmeans.fit(dataset)

# Evaluate clustering by computing Within Set Sum of Squared Errors.
wssse = model.computeCost(dataset)
print("Within Set Sum of Squared Errors = " + str(wssse))

# Shows the result.
centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

Kmeans

```
+-----+--------------------------+
|label|features                  |
+-----+--------------------------+
|0.0  |(3,[],[])                 |
|1.0  |(3,[0,1,2],[0.1,0.1,0.1]) |
|2.0  |(3,[0,1,2],[0.2,0.2,0.2]) |
|3.0  |(3,[0,1,2],[9.0,9.0,9.0]) |
|4.0  |(3,[0,1,2],[9.1,9.1,9.1]) |
|5.0  |(3,[0,1,2],[9.2,9.2,9.2]) |
+-----+--------------------------+

Within Set Sum of Squared Errors = 0.12
Cluster Centers:
[ 0.1  0.1  0.1]
[ 9.1  9.1  9.1]
```

ALS code part 1

```
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.recommendation import ALS
from pyspark.sql import Row
import urllib2

html = urllib2.urlopen('https://raw.githubusercontent.com/apache/spark/branch-
2.1/data/mllib/als/sample_movielens_ratings.txt') .read()
text_file = open("sample_movielens_ratings.txt", "w")
text_file.write(html)
text_file.close()

lines = spark.read.text("sample_movielens_ratings.txt").rdd
parts = lines.map(lambda row: row.value.split("::"))
ratingsRDD = parts.map(lambda p: Row(userId=int(p[0]), movieId=int(p[1]),
                                     rating=float(p[2]), timestamp=long(p[3])))
ratings = spark.createDataFrame(ratingsRDD)
```

ALS code part 2

```
print("Total count of movie ratings is " + str(ratings.count()))
ratings.sample(fraction=0.01, withReplacement=False).show()
(training, test) = ratings.randomSplit([0.8, 0.2])

# Build the recommendation model using ALS on the training data
als = ALS(maxIter=5, regParam=0.01, userCol="userId", itemCol="movieId", ratingCol="rating")
model = als.fit(training)

# Evaluate the model by computing the RMSE on the test data
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse", labelCol="rating",
                                predictionCol="prediction")
rmse = evaluator.evaluate(predictions)
print("Root-mean-square error = " + str(rmse))
```

ALS Output

```
Total count of movie ratings is 1501
+-------+------+----------+------+
|movieId|rating| timestamp|userId|
+-------+------+----------+------+
|      2|   2.0|1424380312|     1|
|     21|   3.0|1424380312|     1|
|     45|   1.0|1424380312|     6|
|     77|   1.0|1424380312|     6|
|     89|   1.0|1424380312|    11|
|     74|   5.0|1424380312|    22|
|     46|   1.0|1424380312|    24|
|     85|   1.0|1424380312|    29|
+-------+------+----------+------+

Root-mean-square error = 1.71474156957
```

TUNING

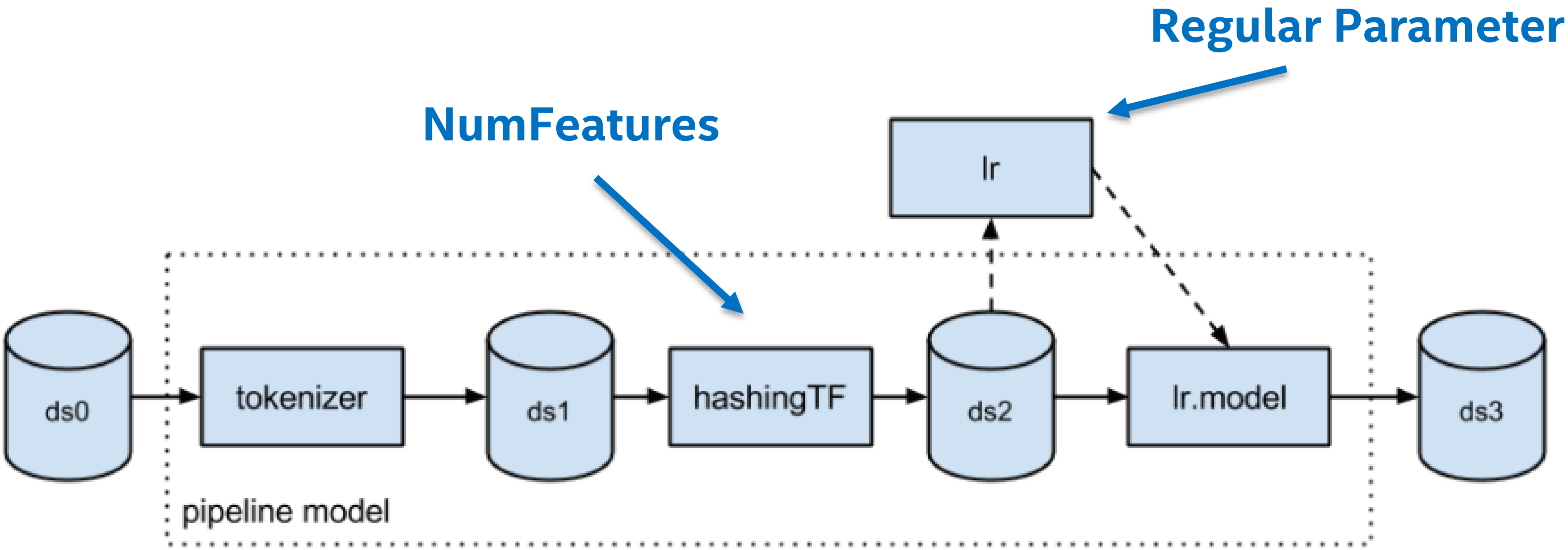
Give a man a fish and you feed him for a day. Teach a man to fish and you feed him for a lifetime. - Lao Tzu

Tuning

We will take a look at

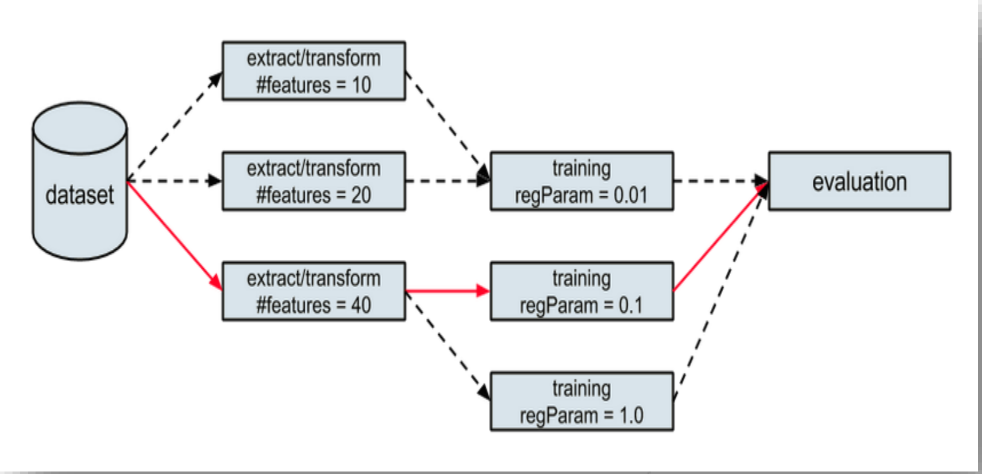
- Hyper-parameter tuning via cross validation
- Native library speedup machine learning

How to choose hyper parameter



Hyper-parameter tuning via cross validation

```
// Build a parameter grid.
val paramGrid = new ParamGridBuilder()
  .addGrid(hashingTF.numFeatures, Array(10, 20, 40))
  .addGrid(lr.regParam, Array(0.01, 0.1, 1.0))
  .build()
// Set up cross-validation.
val cv = new CrossValidator()
  .setNumFolds(3)
  .setEstimator(pipeline)
  .setEstimatorParamMaps(paramGrid)
  .setEvaluator(new BinaryClassificationEvaluator)
// Fit a model with cross-validation.
val cvModel = cv.fit(trainingDataset)
```



Native library speedup machine learning

Intel® Math Kernel Library, fastest math kernel implementation on Intel Architecture.

Order of magnitude than JVM implementation

- Linear Algebra (BLAS, sparse BLAS)
- FFT
- Vector Math
- Statistic

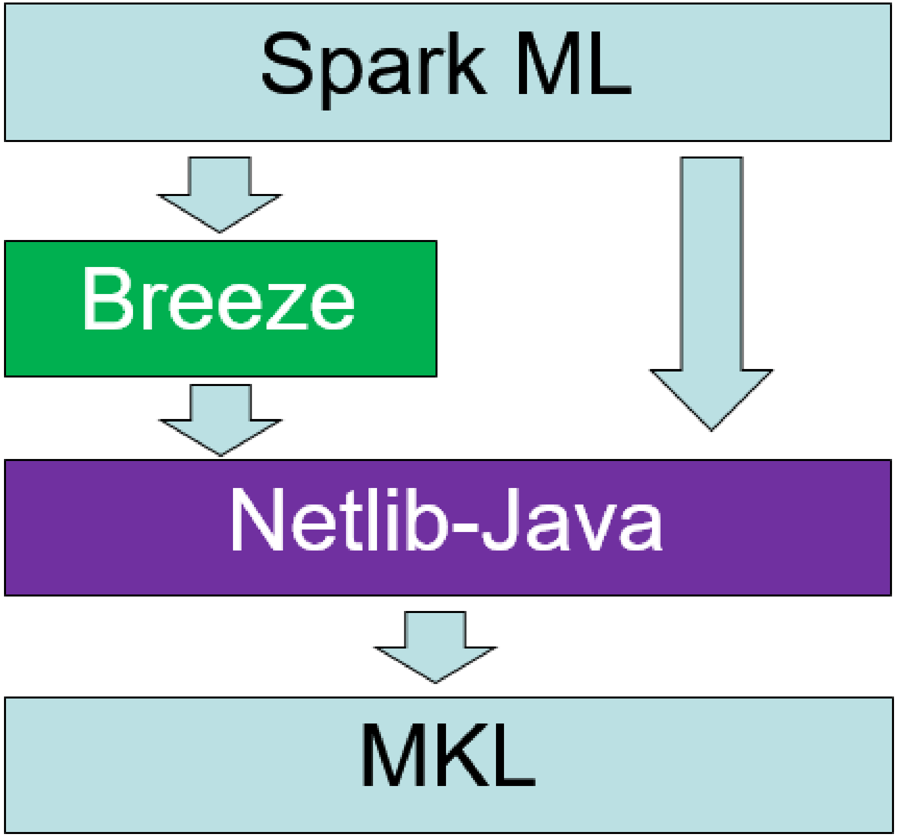
It Free!

https://software.intel.com/en-us/mkl

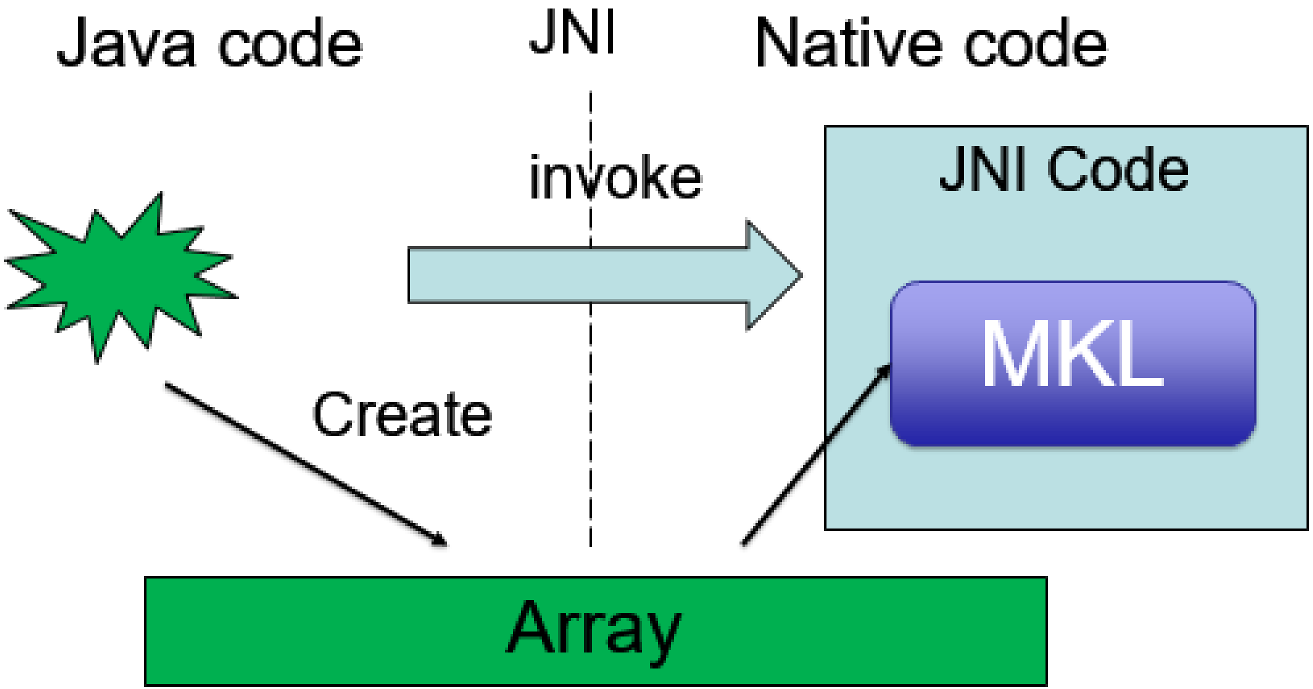
How to speed up your application via MKL

Netlib-Java (JNI)

- routines invocation trigger class load
- extract so files in Jar to a tmp file
- JVM load that so file
- OS load so file dependency
- if load succeed, use routine implemented in native local so file, or roll-back to JVM version routine



How does it work?



Some Pitfalls

You need notice that

- you should install MKL on each of your machine, and link the MKL so files correctly, see netlib-java doc
- you need to recompile spark with a -P **netlib**-lpgp
- set OMP_THREAD_NUM careful
- don't exceed the physical core number

TAKE A BREAK

intel®

Software

Legal Disclaimer

Risk Factors

The above statements and any others in this document that refer to plans and expectations for the first quarter, the year and the future are forward-looking statements that involve a number of risks and uncertainties. Words such as “anticipates,” “expects,” “intends,” “plans,” “believes,” “seeks,” “estimates,” “may,” “will,” “should” and their variations identify forward-looking statements. Statements that refer to or are based on projections, uncertain events or assumptions also identify forward-looking statements. Many factors could affect Intel’s actual results, and variances from Intel’s current expectations regarding such factors could cause actual results to differ materially from those expressed in these forward-looking statements. Intel presently considers the following to be the important factors that could cause actual results to differ materially from the company’s expectations. Demand could be different from Intel's expectations due to factors including changes in business and economic conditions; customer acceptance of Intel’s and competitors’ products; supply constraints and other disruptions affecting customers; changes in customer order patterns including order cancellations; and changes in the level of inventory at customers. Uncertainty in global economic and financial conditions poses a risk that consumers and businesses may defer purchases in response to negative financial events, which could negatively affect product demand and other related matters. Intel operates in intensely competitive industries that are characterized by a high percentage of costs that are fixed or difficult to reduce in the short term and product demand that is highly variable and difficult to forecast. Revenue and the gross margin percentage are affected by the timing of Intel product introductions and the demand for and market acceptance of Intel's products; actions taken by Intel's competitors, including product offerings and introductions, marketing programs and pricing pressures and Intel’s response to such actions; and Intel’s ability to respond quickly to technological developments and to incorporate new features into its products. The gross margin percentage could vary significantly from expectations based on capacity utilization; variations in inventory valuation, including variations related to the timing of qualifying products for sale; changes in revenue levels; segment product mix; the timing and execution of the manufacturing ramp and associated costs; start-up costs; excess or obsolete inventory; changes in unit costs; defects or disruptions in the supply of materials or resources; product manufacturing quality/yields; and impairments of long-lived assets, including manufacturing, assembly/test and intangible assets. Intel's results could be affected by adverse economic, social, political and physical/infrastructure conditions in countries where Intel, its customers or its suppliers operate, including military conflict and other security risks, natural disasters, infrastructure disruptions, health concerns and fluctuations in currency exchange rates. Expenses, particularly certain marketing and compensation expenses, as well as restructuring and asset impairment charges, vary depending on the level of demand for Intel's products and the level of revenue and profits. Intel’s results could be affected by the timing of closing of acquisitions and divestitures. Intel's results could be affected by adverse effects associated with product defects and errata (deviations from published specifications), and by litigation or regulatory matters involving intellectual property, stockholder, consumer, antitrust, disclosure and other issues, such as the litigation and regulatory matters described in Intel's SEC reports. An unfavorable ruling could include monetary damages or an injunction prohibiting Intel from manufacturing or selling one or more products, precluding particular business practices, impacting Intel’s ability to design its products, or requiring other remedies such as compulsory licensing of intellectual property. A detailed discussion of these and other factors that could affect Intel’s results is included in Intel’s SEC filings, including the company’s most recent reports on Form 10-Q, Form 10-K and earnings release.