Schnelles Denken - Maschinelles Lernen mit Apache Spark 2

Heiko Spindler

What is Apache Spark?

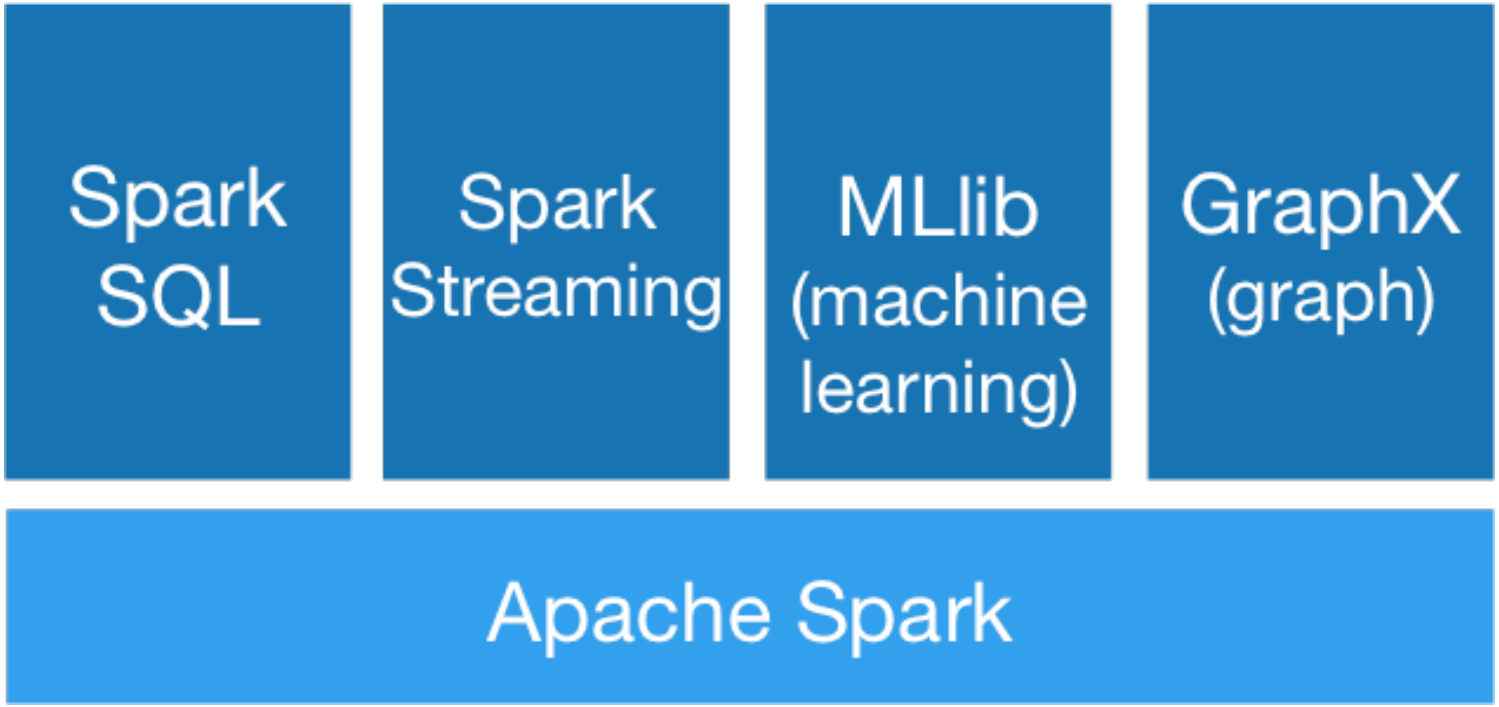
Apache Spark is an distributed open source big data processing framework built around speed, ease of use, and sophisticated analytics.

It was originally developed in 2009 in UC Berkeley's AMPLab, and open sourced in 2010 as an Apache project.

Apache Spark – Design Targets

Speed
Ease of Use
Generality
Runs Everywhere
Distribution
Implemented with Scala
API for Scala, Python, Java, R

Apache Spark - Components



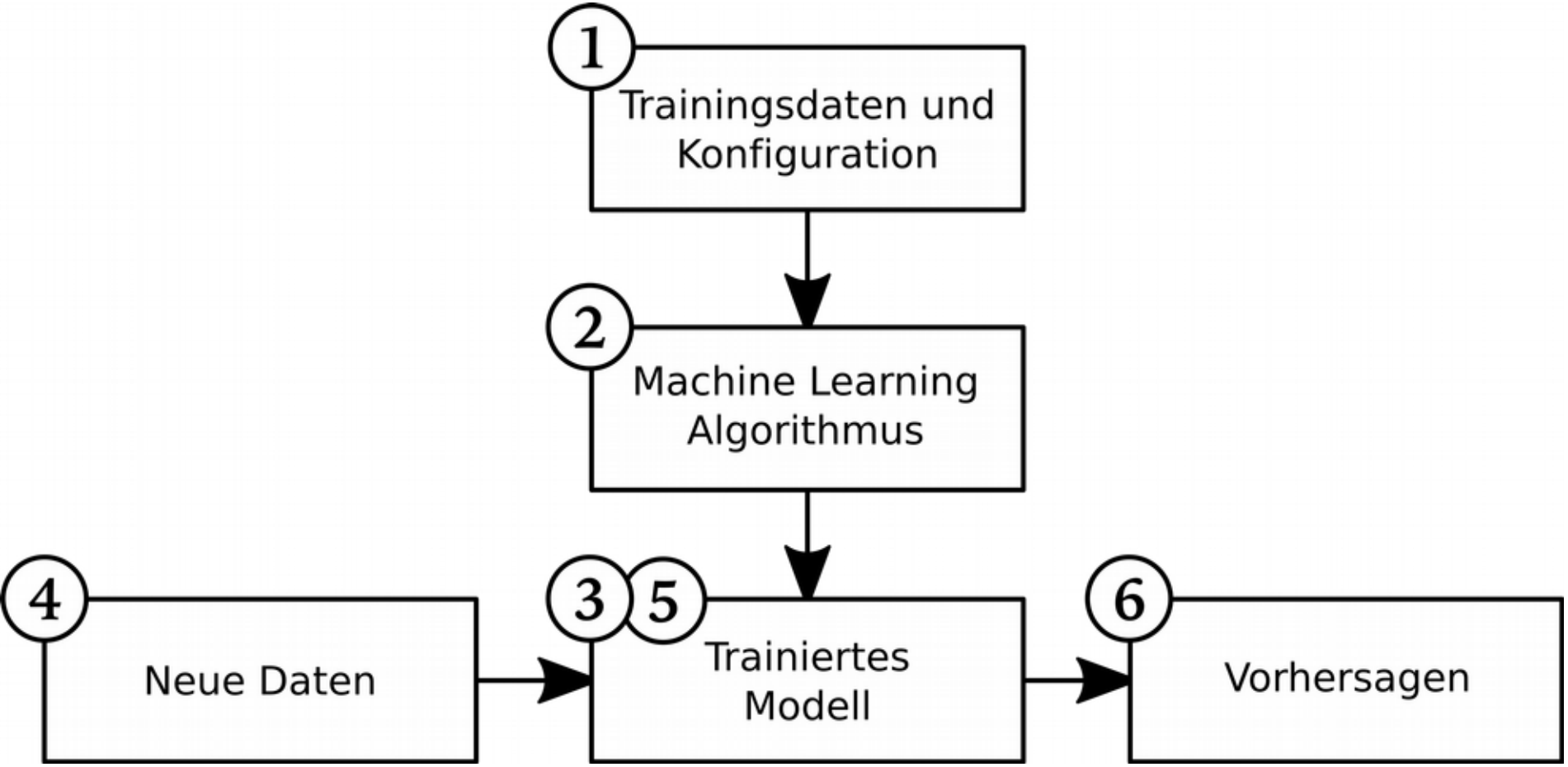
Machine Learning

Machine learning explores the construction and study of algorithms that can learn from data.

These algorithms build models based on inputs and make predictions or decisions, rather than following explicitly programmed instructions.

Spark ML Algorithmen

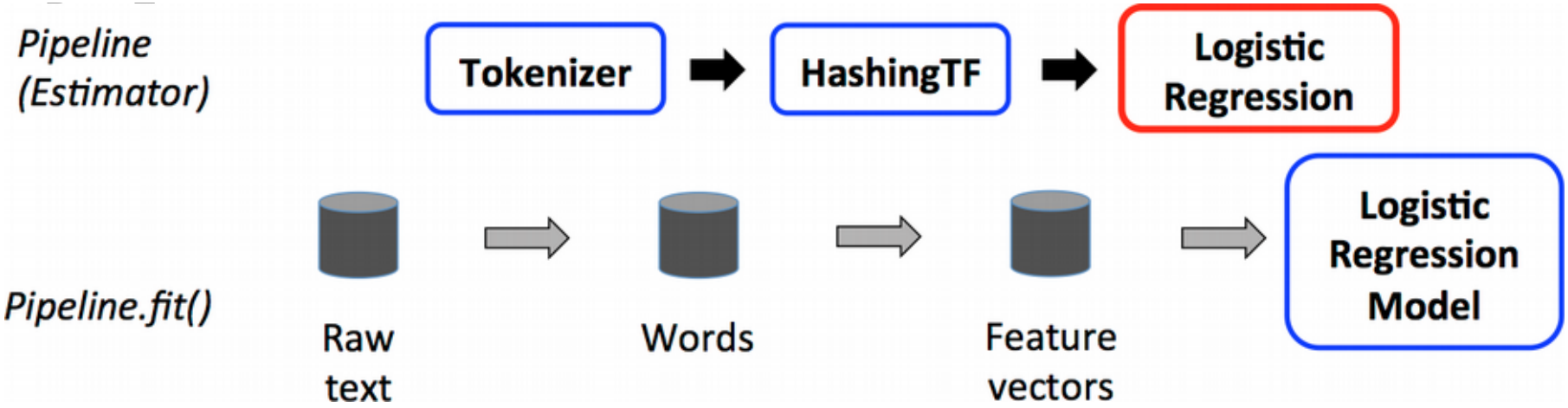
- Feature transformations: standardization, normalization, hashing,...
- Classification: logistic regression, naive Bayes,...
- Regression: generalized linear regression, isotonic regression,...
- Decision trees, random forests, and gradient-boosted trees
- Recommendation: alternating least squares (ALS)
- Clustering: K-means, Gaussian mixtures (GMMs),...
- Topic modeling: latent Dirichlet allocation (LDA)
- Model evaluation and hyper-parameter tuning
- Survival analysis: accelerated failure time model
- Sequential pattern mining: FP-growth, association rules, PrefixSpan
- Distributed linear algebra: singular value decomposition (SVD), principal component analysis (PCA),...
- Statistics: summary statistics, hypothesis testing,...


1
Trainingsdaten und Konfiguration
2
Machine Learning Algorithmus
4
Neue Daten
3
5
Trainiertes Modell
6
Vorhersagen

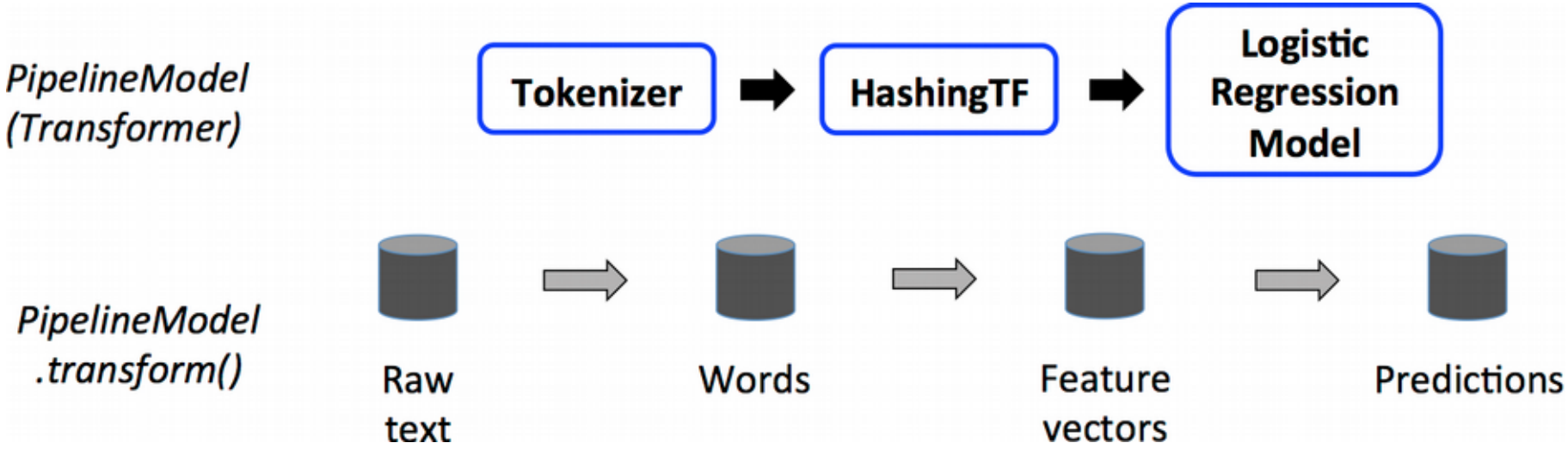
Spark ML – Pipline Elemente

- Extractors
- Transformers + Selectors
- Estimators

Basics – Pipelines (Training)



Basics – Pipelines (Production)



ML Extractors – Transformers

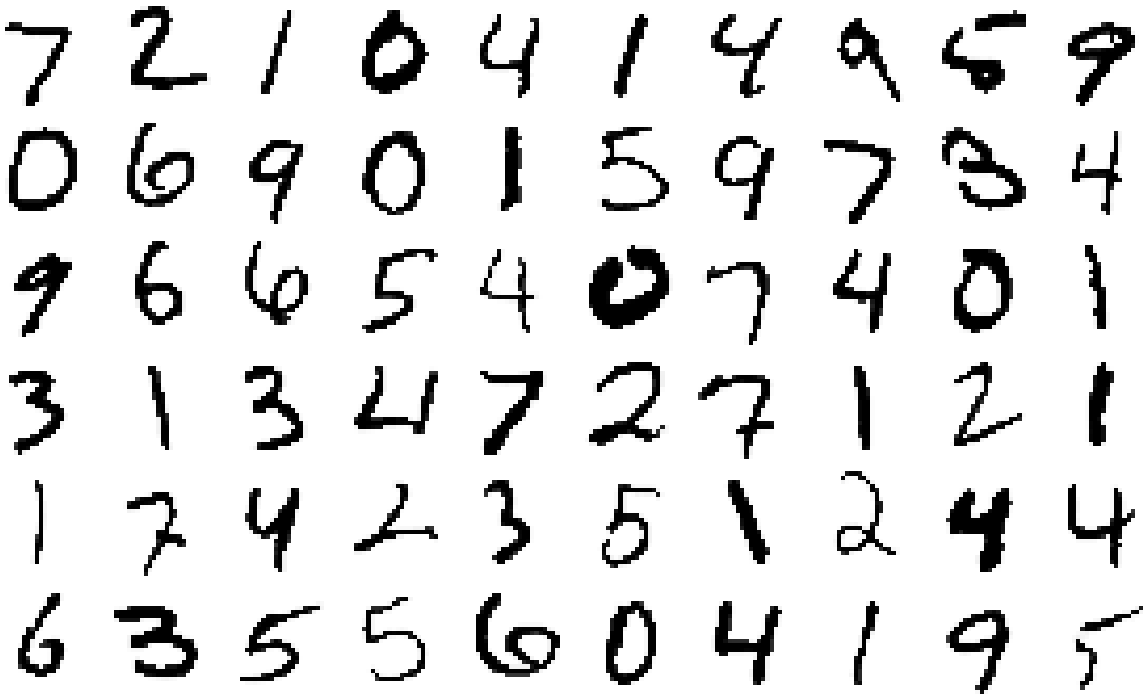
Tokenizer StopWordsRemover nn-gram Binarizer PCA PolynomialExpansion Discrete Cosine Transform (DCT)	StringIndexer IndexToString OneHotEncoder VectorIndexer Normalizer StandardScaler MinMaxScaler MaxAbsScaler	Bucketizer ElementwiseProduct SQLTransformer VectorAssembler QuantileDiscretizer

MNIST – Aufgabenstellung

Bietet Test- und Trainingsdaten für die Erkennung bzw. Klassifizierung von handgeschriebenen Zeichen (0-9).
50.000 Trainingsdaten mit Klassifizierung
10.000 Testdaten mit Klassifizierung

Auf der Webseite sind mehrere Lösungsansätze mit Vorgehen und Ergebnissen verlinkt.

MNIST – Beispiele der Ziffern



MNIST - Online

http://myselph.de/neuralNet.html

MNIST - Daten

Datenformat ist auf der Webseite dokumentiert.
Es gibt Skripte, die z.B. CSV erzeugen.

Label plus Pixeldaten:
Komma-separiert für alle 28 x 28 Pixel: 0: Schwarz, 255: Weiss

Header:
Label, p0, p1, … p783

MNIST – Daten Beispiel

2,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
89,207,253,255,206,88,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
0,0,0,0,0,0,0,0,76,215,252,252,253,252,246,122,0
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,225,239,1
80,55,119,246,252,230,25,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,225,195,0,0,0,122,246,252,112,0,0
,0,0,0,0,0,0,0...

Schritte

- Einlesen der Daten
- Vorbereiten der Daten: Zerlegen in Label und Feature-Vektor
- Konfigurieren des ML Algorithmus
- Trainieren des Models
- Prüfen des Models mit Testdaten
- Auswertung

Vorbereitung

Spark konfigurieren

```
SparkSession spark = SparkSession
        .builder()
        .appName("JavaMNISTDT")
        .master("local[3]")
        .getOrCreate();
```

CSV Einlesen

```
DataFrameReader reader = spark.read()
        .option("header", "true")
        .option("delimiter", ",")
        .option("inferSchema", true)
        .format("com.databricks.spark.csv");

Dataset<Row> test = reader
        .load(Const.BASE_DIR_DATASETS+"mnist_test2.csv")
        .filter(e ->  Math.random() > 0.00 );
Dataset<Row> train = reader
        .load(Const.BASE_DIR_DATASETS+"mnist_train2.csv")
        .filter(e ->  Math.random() > 0.00 );
```

Feature Vector erzeugen

```
StructType schema = train.schema();

String[] inputFeatures =
    Arrays.copyOfRange(schema.fieldNames(), 1,
    schema.fieldNames().length);

VectorAssembler assembler = new VectorAssembler()
        .setInputCols( inputFeatures )
        .setOutputCol( "features" );
```

StringIndexer erzeugen

```
StringIndexerModel stringIndexer = new StringIndexer()
        .setInputCol("label")
        .setHandleInvalid("skip")
        .setOutputCol("indexedLabel")
        .fit(train);
```

Bildet die Labels als numerische Werte ab.

Entscheidungsbaum

DT erzeugen

```
DecisionTreeClassifier dt = new DecisionTreeClassifier()
    .setMaxDepth(28)
    .setSeed(12345L)
    .setLabelCol(stringIndexer.getOutputCol())
    .setFeaturesCol(assembler.getOutputCol());
```

IndexToString

```
IndexToString indexToString = new IndexToString()
        .setInputCol("prediction")
        .setOutputCol("predictedLabel")
        .setLabels(stringIndexer.labels());
```

Rücktransformation der numerischen Labels auf Sprechende / fachliche Bezeichnungen

Pipeline

```
Pipeline pipeline = new Pipeline()
        .setStages(new PipelineStage[] {
                assembler
                , stringIndexer
                , dt
                , indexToString
        });
```

Führt die Schritte in einem Ablauf zusammen

Ausführen der Pipeline

```
PipelineModel model = pipeline.fit(train);
// Use the model to evaluate the test set.
Dataset<Row> result = model.transform(test);
```

Führt die Schritte in einem Ablauf zusammen

Auswertung

```
String correct
= "Correct:"+ (100.0 * result.filter("label =
predictedLabel").count() / result.count());
```

Berechnen des Anteils korrekt erkannter Ziffern

MNIST – Ergebnisse DT

label	0	1	2	3	4	5	6	7	8	9
0	919	1	12	5	3	6	12	3	7	12
1	1	1092	6	7	1	5	5	2	15	1
2	14	6	881	33	12	9	16	31	24	6
3	8	5	34	852	8	44	2	14	24	19
4	5	2	8	3	859	3	19	20	20	43
5	11	9	6	44	10	748	12	8	24	20
6	18	7	12	2	16	21	859	1	17	5
7	2	15	20	19	8	8	1	922	11	22
8	23	3	25	31	16	25	12	19	790	30
9	11	2	6	17	27	15	1	17	29	884

Duration: ~2 minutes
Precision: 0.8806

Random Forests

RandomForest

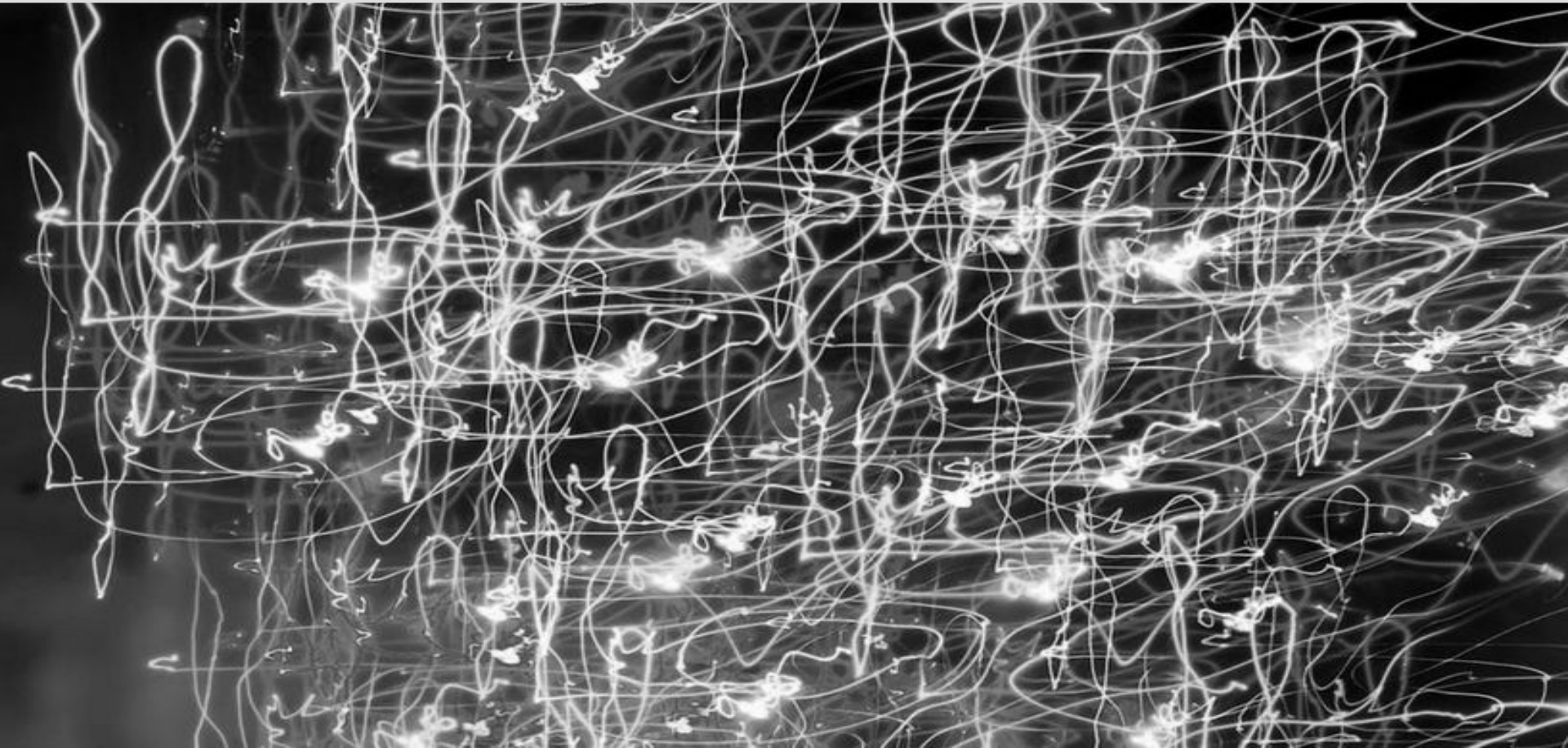
```
RandomForestClassifier rf = new RandomForestClassifier()
        .setNumTrees(30)
        .setLabelCol("indexedLabel")
        .setFeaturesCol(assembler.getOutputCol());

Pipeline pipeline = new Pipeline()
        .setStages(new PipelineStage[] {
                assembler
                , stringIndexer
                , rf
                , indexToString
        });
```

MNIST – Ergebnisse RF

label	0	1	2	3	4	5	6	7	8	9
0	968		1			3	4	1	3	
1		1122	4	3	1		4		1	
2	8		998	5	4	1	3	7	4	2
3			8	974		7	1	9	5	6
4	1	1	3	1	940		6	1	8	21
5	3		3	17	3	840	5	5	12	4
6	9	3	3	1	5	7	926		4	
7	1	5	20	2	1			984	1	14
8	2		7	12	5	4	4	4	926	10
9	7	5	4	11	14	4	1	4	6	953

Duration: ~4 minutes
Precision: 0.963

Neuronale Netze



MultiLayerPerceptronClassifier (MLPC)

784 Pixel => Input Neuronen
800 Versteckte Neuronen
10 Ausgabe Neuronen => 10 Zielklassen (10 Ziffern)

Alle Neuronen sind mit allen Neuronen der vorherigen Schicht verbunden

MNIST - MLPC

```
int[] layers = { 784, 800, 100, 10 };

MultilayerPerceptronClassifier mlpc = new
MultilayerPerceptronClassifier()
        .setLabelCol(stringIndexer.getOutputCol())
        .setFeaturesCol(assembler.getOutputCol())
        .setLayers(layers);
```

MNIST - MLPC

```
int[] layers = { 784, 800, 100, 10 };

MultilayerPerceptronClassifier mlpc = new
MultilayerPerceptronClassifier()
       .setLabelCol(stringIndexer.getOutputCol())
       .setFeaturesCol(assembler.getOutputCol())
       .setLayers(layers);
```

Laufzeit: 33 Minuten
Ergebnisse verbessert sich leicht auf 96,7%

MNIST – Ergebnisse MLPC

label	0	1	2	3	4	5	6	7	8	9
0	970		1			3	2	2	1	1
1		1128	1	1		1	3		1	
2	5	1	1003	6	2	1	3	6	5	
3			4	987		7	1	4	3	4
4	1		2		958		5	4		12
5	3	1		6	3	869	2	1	4	3
6	4	3	3		7	4	936		1	
7		3	6	3	1			1009		6
8	2	1	3	6	4	4	1	4	945	4
9	3	4			10	3	3	6	5	975

Duration: ~63 minutes
Precision: 0.978

MNIST - Optimierung

- Binarizer
- Affine-Transformationen (z.B. Drehungen + / - 10 Grad)
- ...

Links

Heise.de Developer:
https://heise.de/-3657735

SourceCode:
https://github.com/brainbrix/spark-tests

Links

http://pjreddie.com/projects/mnist-in-csv/
http://yann.lecun.com/exdb/mnist/
http://mike.seddon.ca/using-apache-spark-neural-networks-to-recognise-digits/…

Fazit

Es gibt mehr ML Algorithmen als nur Neuronale Netze!
Spark bringt viele Standard ML Algorithmen mit.
Mächtiges Pipeline-Konzept (Modell-Persistenz und Optimierung)

Heiko Spindler
hs@heikospindler.de