Zurich University
of Applied Sciences

Information Engineering 2

Machine Learning with Spark

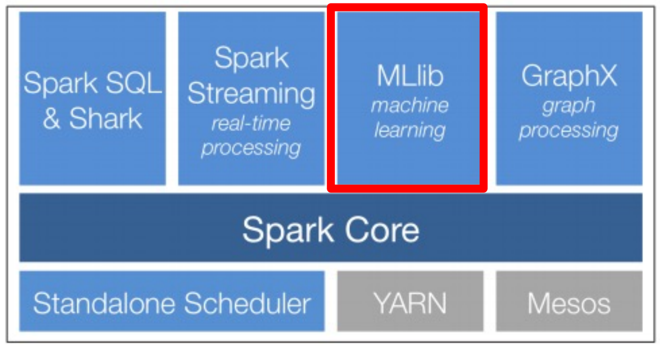
Prof. Dr. Kurt Stockinger

Semesterplan

SW	Datum	Vorlesungsthema	Praktikum
1	23.02.2022	Data Warehousing Einführung	Praktikum 1: KNIME Tutorial
2	02.03.2022	Dimensionale Datenmodellierung 1	Praktikum 1: KNIME Tutorial (Vertiefung)
3	09.03.2022	Dimensionale Datenmodellierung 2	Praktikum 2: Datenmodellierung
4	16.03.2022	Datenqualität und Data Matching	Praktikum 3: Star-Schema, Bonus: Praktikum 4: Slowly Changing Dimensions
5	23.03.2022	Big Data Einführung	DWH Projekt - Teil 1
6	30.03.2022	Spark - Data Frames	DWH Projekt - Teil 2 (Abgabe: 4.4.2022 23:59:59)
7	06.04.2022	Data Storage: Hadoop Distributed File System & Parquet	Praktikum 1: Data Frames
8	13.04.2022	Query Optimization	Praktikum 2: Data Storage
9	20.04.2022	Spark Best Practices & Applications	Praktikum 3: Query Optimization & Performance Analysis
10	27.04.2022	Machine Learning mit Spark 1	Praktikum 3: Query Optimization & Performance Analysis (Vertiefung)
11	04.05.2022	Machine Learning mit Spark 2 + Q&A	Praktikum 4: Machine Learning (Regression)
12	11.05.2022	NoSQL Systems	Big Data Projekt - Teil 1
13	18.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 2
14	25.05.2022	Keine Vorlesung (Arbeit am Projekt)	Big Data Projekt - Teil 3 (Abgabe: 30.5.2022 23:59:59)

Educational Objectives for Today

- Understand the difference between supervised and unsupervised machine learning:
  - Linear and logistic regression

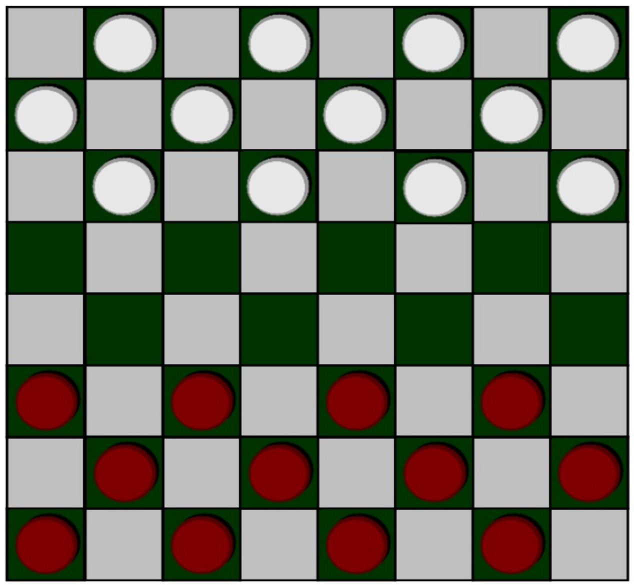


- Learn about main concepts of Spark ML
- Understand machine learning pipelines
- Apply logistic regression using machine learning pipelines
- Scalable Machine Learning:
  - Understand importance of sparseness of vectors and matrices
  - Understand major performance optimization possibilities for machine learning

Machine Learning Definition

Zurich University
of Applied Sciences

Arthur Samuel (1959). Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed.



Supervised vs. Unsupervised Learning

- Supervised Learning:
  - Given some training data with labels, predict labels of new data
  - E.g.
    - Email 1: spam, Email 2: spam, Email 3: not spam, etc.
    - New email: spam / not spam?
  - Algorithms: Logistic regression, Support Vector Machines, Neural Networks

- Unsupervised Learning:
  - Given a set of unlabeled data points, find some commonalities or structure
  - E.g.
    - Information about people who buy a house (age, income, occupation,…)
    - Classify people into 5 groups
  - Algorithms: k-means Clustering

Linear Regression

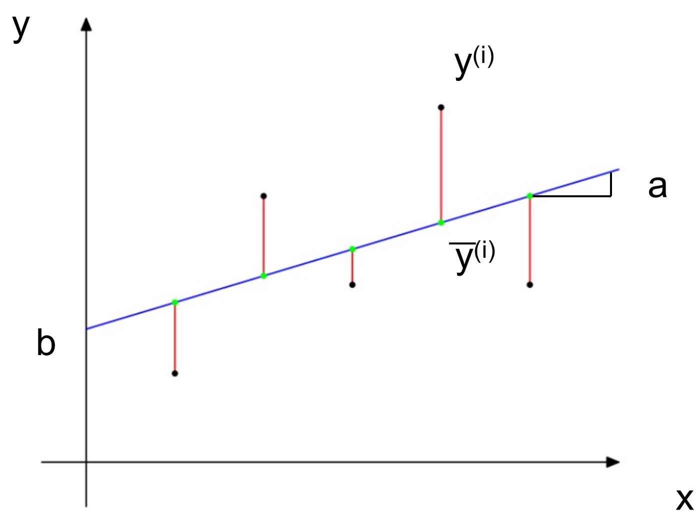
- Given:
  - Information about people: size + shoe size

- How do we estimate the size of a person given the shoe size?

Goal of Linear Regression

- y = ax + b … a = slope, b = y-interceptor
- Find parameters a and b of linear function that minimizes the mean squared error (MSE)

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(\hat{y}^{(i)} - y^{(i)})^2$$



Logistic Regression

- Extension of linear regression
- Used for non-linear data (when fitting a simple “line” is not enough)
- One of the most commonly used supervised machine learning algorithms
- Serves as the basis for more complex algorithms such as neural networks

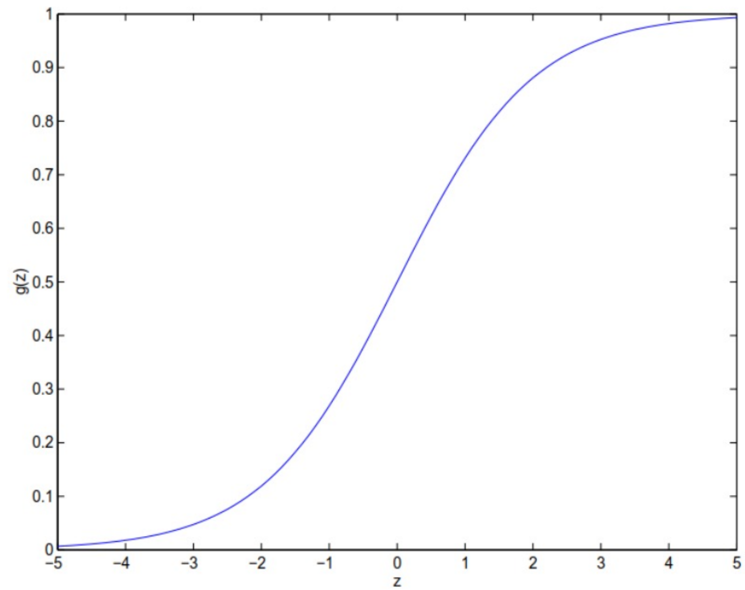
Logistic Regression: Binary Classification Task

- Given:
  - Information about tumor size
- Goal:
  - Is the tumor malignant or not?

Logistic Regression: Fitting Function

$$h_\theta(x) = g(\theta^T x) = \frac{1}{1 + e^{-\theta^T x}},$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

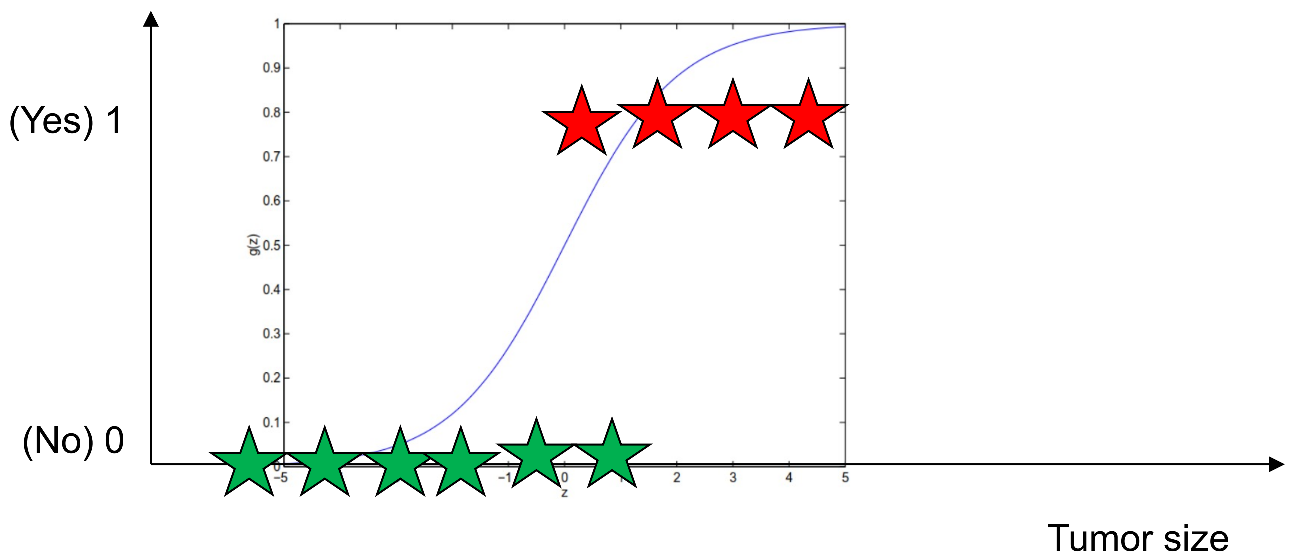


"Logistic Function" or "Sigmoid Function"

→ Distributes smoothly between 0 and 1

Logistic Regression - Revisited
Binary Classification Task

- Given:
  - Information about tumor size
- Goal:
  - Is the tumor malignant or not?



Linear Regression with Multiple Features – A Vector-Multiplication Problem

- Regression equation: $\hat{y} = b_0 + b_1x_1 + b_2x_2$
- $\hat{y}$ is the predicted value; b_0, b_1, and b_2 are regression coefficients;

person	score	IQ	study hours
1	100	110	40
2	90	120	30
3	80	100	20
4	70	90	0
5	60	80	10

Linear Regression with Multiple Features – A Vector-Matrix Multiplication Problem

- Regression coefficient can be expressed with the following equation:
  - $b = (X'X)^{-1}X'Y$
- Now we have a vector-matrix multiplication problem which can be parallelized
- Good for Big Data computations

Machine Learning Objectives

- Many machine learning problems can be formulated as a convex optimization problem of the form

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

where

- x_i … training examples (features)
- y_i … corresponding labels that need to be predicted
- w … weights for features (vector of dimension d)
- g … gradient of the loss
- n … #training points (can be large!)

- Summation of losses
- Goal: find the best w with optimization, i.e. minimize error on training data
- Method is linear if the function can be expressed as $w^T x$ and y
- In summary, many machine learning problems rely on efficient linear algebra libraries

Parallel Gradient Descent Example

Vector Update: $\mathbf{w}_{i+1} = \mathbf{w}_i - \alpha_i \sum_{j=1}^{n} \underline{(\mathbf{w}_i^\top \mathbf{x}^{(j)} - y^{(j)})\mathbf{x}^{(j)}}$

Compute summands in parallel!
note: workers must all have $\mathbf{w}_i$

Example: n = 6; 3 workers

$\mathbf{w}_{i+1}$

workers:	— $\mathbf{x}^{(1)}$ — — $\mathbf{x}^{(5)}$ —	— $\mathbf{x}^{(3)}$ — — $\mathbf{x}^{(4)}$ —	— $\mathbf{x}^{(2)}$ — — $\mathbf{x}^{(6)}$ —	O(nd) Distributed Storage	
map:	$(\mathbf{w}_i^\top \mathbf{x}^{(j)} - y^{(j)})\mathbf{x}^{(j)}$	$(\mathbf{w}_i^\top \mathbf{x}^{(j)} - y^{(j)})\mathbf{x}^{(j)}$	$(\mathbf{w}_i^\top \mathbf{x}^{(j)} - y^{(j)})\mathbf{x}^{(j)}$	O(nd) Distributed Computation	O(d) Local Storage
reduce:		$\sum_{j=1}^{n} (\mathbf{w}_i^\top \mathbf{x}^{(j)} - y^{(j)})\mathbf{x}^{(j)}$		O(d) Local Computation	O(d) Local Storage

n… #data points

d … #features

(from Ameet Talwalkar, UCLA)

Gradient Descent Summary

- Can easily be parallelized
- Each iteration is “cheap”
- Stochastic variants can make computation even “cheaper”

MLlib: Scalable Machine Learning Library

- Machine learning for Java, Scala, Python and R
- Interoperates with NumPy in Python
- Supports various data sources:
  - Text files
  - HDFS (Hadoop distributed file system)
- Allows plugging in Hadoop workflows
- Contains rich selection of distributed data structures and algorithms

Major Algorithms Supported by MLlib

- Classification and regression:
  - Linear models (SVM, linear regression, logistic regression)
  - Naïve Bays
  - Decision trees
  - Ensemble trees

- Collaborative filtering:
  - Alternating least squares

- Clustering:
  - K-means
  - Gaussian mixtures

- Dimensionality reduction:
  - Singular value decomposition (SVD)
  - Principle component analysis (PCA)

Concepts of Spark ML
Machine Learning Pipelines

- Generalized API for training and tuning ML algorithms

- Pipeline functionality:
  - Sequences of ML algorithms are treated as one unit

MLlib standardizes APIs for machine learning algorithms to make it easier to combine multiple algorithms into a single pipeline, or workflow. This section covers the key concepts introduced by the Pipelines API, where the pipeline concept is mostly inspired by the scikit-learn project.

- **`DataFrame`**: This ML API uses `DataFrame` from Spark SQL as an ML dataset, which can hold a variety of data types. E.g., a `DataFrame` could have different columns storing text, feature vectors, true labels, and predictions.
- **`Transformer`**: A `Transformer` is an algorithm which can transform one `DataFrame` into another `DataFrame`. E.g., an ML model is a `Transformer` which transforms a `DataFrame` with features into a `DataFrame` with predictions.
- **`Estimator`**: An `Estimator` is an algorithm which can be fit on a `DataFrame` to produce a `Transformer`. E.g., a learning algorithm is an `Estimator` which trains on a `DataFrame` and produces a model.
- **`Pipeline`**: A `Pipeline` chains multiple `Transformer`s and `Estimator`s together to specify an ML workflow.
- **`Parameter`**: All `Transformer`s and `Estimator`s now share a common API for specifying parameters.

Transformer

- Abstraction that includes feature transformers and learning models

- Transforms DataFrame to another by appending columns

- Example: Feature transformer:
  - Takes a DataFrame
  - Reads a column (e.g. text)
  - transform()-method: Maps it into a new value (e.g. feature vector)
  - Outputs a new DataFrame with mapped column appended

- Example: Learning model:
  - Takes a DataFrame
  - Reads the column containing feature vectors
  - Predicts the label for each feature vector
  - Outputs a new DataFrame with predicted labels appended as a column

Estimator

- Abstraction for learning algorithm that fits or trains data

- Example:
  - Input:
    - DataFrame
    - Learning algorithm (e.g. logistic regression) = Estimator
  - fit()-function: train logistic regression
  - Output:
    - Trained model = Transformer

Example Pipeline

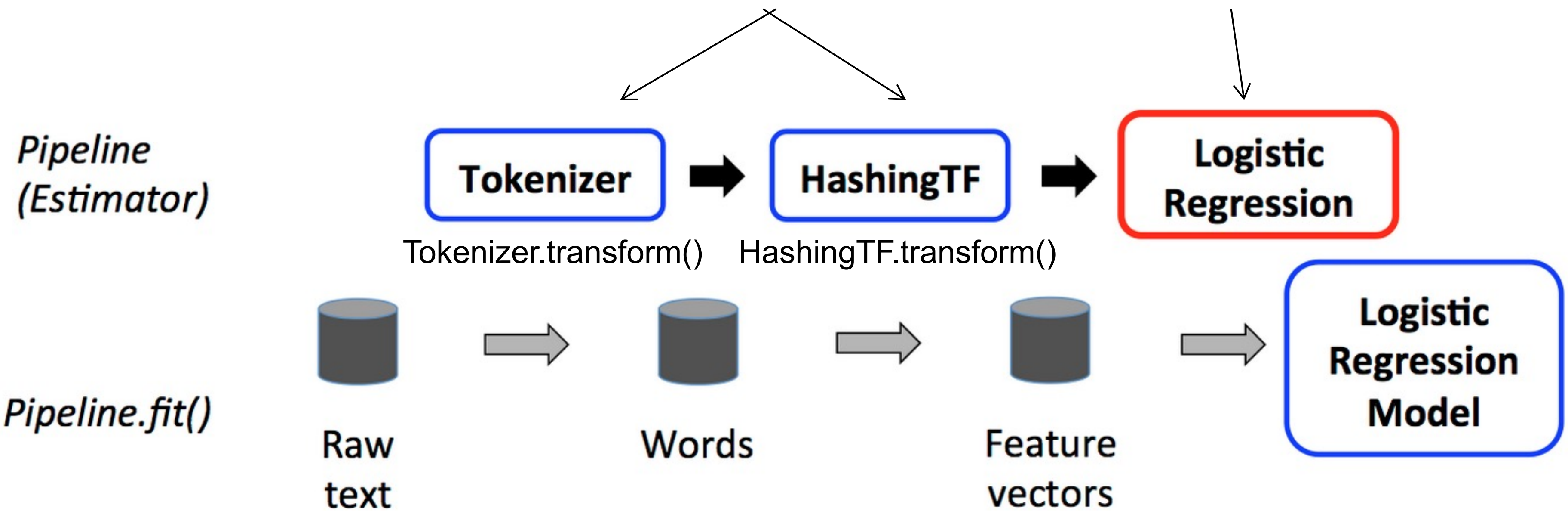
- Machine learning often requires executing a workflow

- Example:
  - Split each document's text into words.
  - Convert each document's words into a numerical feature vector.
  - Learn a prediction model using the feature vectors and labels.

- Workflow = sequence of PipelineStages (Transformers and Estimators) to be run in a specific order

Example: Pipelined Model Training

Top part: Pipeline with 3 stages: 2 transformers (blue), 1 estimator (red)



Bottom part: Data flow – cylinders indicate DataFrames

LogisticRegression.fit() is called on original DataFrame (text with labels)

Example: Logistic Regression Analysis
Simple Pipeline

```
1 0 2.52078447201548 0 0 0 2.004684436494304 2.000347299268
0 2.857738033247042 0 0 2.619965104088255 0 2.0046844364943
0 2.857738033247042 0 2.061393766919624 0 0 2.0046844364943
1 0 0 2.061393766919624 2.619965104088255 0 2.0046844364943
1 2.857738033247042 0 2.061393766919624 2.619965104088255 0
0 2.857738033247042 0 2.061393766919624 2.619965104088255 0
1 0 0 0 2.619965104088255 0 2.004684436494304 0 0 2.2283870
1 0 0 0 2.619965104088255 0 2.004684436494304 0 0 2.2283870
0 2.857738033247042 0 2.061393766919624 2.619965104088255 0
```

```python
from pyspark.ml import Pipeline
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.classification import LogisticRegression

# Load and split data into training and testing sets
df = spark.read.csv("/FileStore/tables/9wnqm5n41489737284042/data_regression.txt", sep=" ", inferSchema=True)

# Rename first column to "label"
df = df.withColumnRenamed(df.columns[0], 'label')

# Split into training and test data
training, testing = df.randomSplit([0.6, 0.4], seed=42)

# Configure an ML pipeline, which consists of two stages: feature assembler and lr.
# Transform n feature vectors into one single vector column
assembler = VectorAssembler(inputCols=training.columns[1:], outputCol='features')
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[assembler, lr])
print "LogisticRegression parameters:\n" + lr.explainParams() + "\n"

# predict
model = pipeline.fit(training)
prediction = model.transform(testing)
```

Potential Output

```
#print "prediction-schema: ", prediction.printSchema()
selected = prediction.select("features", "label", "probability", "prediction")
```

```
+---------+-----+--------------------+----------+
| features|label|         probability|prediction|
+---------+-----+--------------------+----------+
|[4.6,1.5]|    0|[0.99491845702985...|       0.0|
|[4.8,1.8]|    0|[0.99837590339846...|       0.0|
|[4.5,1.5]|    0|[0.99384878230698...|       0.0|
|[4.7,1.5]|    0|[0.99580290469472...|       0.0|
|[4.3,1.3]|    0|[0.98513993446252...|       0.0|
|[4.1,1.3]|    0|[0.97832928702830...|       0.0|
|[4.4,1.4]|    0|[0.99043013874804...|       0.0|
|[4.4,1.2]|    0|[0.98421700830157...|       0.0|
|[4.5,1.5]|    0|[0.99384878230698...|       0.0|
|[4.4,1.3]|    0|[0.98770531185225...|       0.0|
|[3.3,1.0]|    0|[0.81951829651619...|       0.0|
|[4.2,1.2]|    0|[0.97699324497930...|       0.0|
|[3.6,1.3]|    0|[0.94528406269604...|       0.0|
|[5.0,1.7]|    0|[0.99857491290381...|       0.0|
|[4.0,1.3]|    0|[0.97385931051166...|       0.0|
|[4.7,0.1]|    0|[0.87246224604569...|       0.0|
|[3.0,0.2]|    0|[0.25167975227478...|       1.0|
|[1.3,0.3]|    1|[0.01626639049754...|       1.0|
```

Assumption: Only two columns are used as features

Vectors and Matrices

- How would you store this vector?

0, 0, 0, 0, 0, 7, 0, 0, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 6, 0, 0, 0, 0, 0

- … and this matrix ?

0, 0, 0, 0, 0, 7, 0, 0, 0, 0
0, 0, 3, 0, 0, 0, 0, 0, 2, 0
0, 0, 0, 0, 6, 0, 0, 0, 0, 0

Why is Data Sparseness Important?

One-Hot Encoding

- Convert categorical features to numerical
- E.g. country {Argentina, Brazil, …, Switzerland, USA, …}
-

  Sparse vectors:
  - [1, 0, 0, 0, 0, …]
  - [0, 1, 0, 0, 0, …]

- Density: 1/#categories

Xiangrui Meng, Databricks 2013

Netflix Prize (some time ago…)

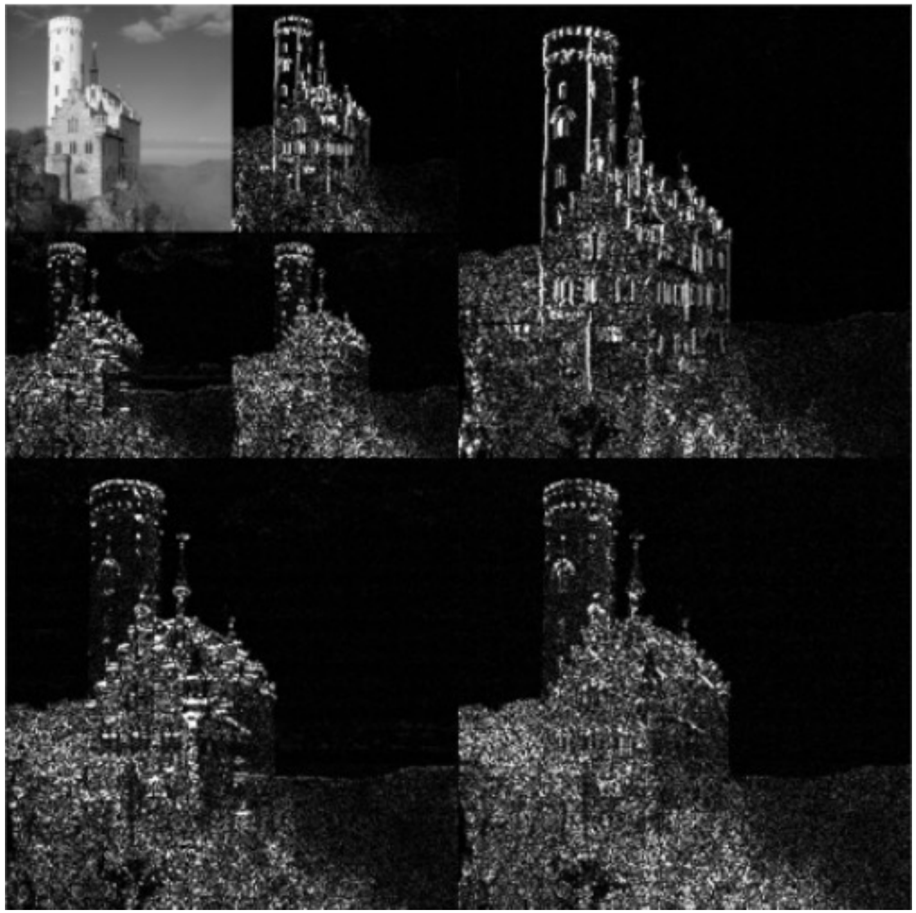
- Number of users: 480,189
- Number of movies: 17,770
- Number of observed ratings: 100,480,507
- Density of user-movie-rating matrix: 1.17%

Xiangrui Meng, Databricks 2013

Images: Dense

Xiangrui Meng, Databricks 2013

Images: Sparse



Wavelet transformations

Xiangrui Meng, Databricks 2013

Exploiting Sparsity

- Spark ML supports sparse input data
- Spark ML takes advantage of sparsity in both storage and computation in
  - Summary statistics
  - Linear methods (linear SVM, logistic regression, …)
  - K-Means
  - Naïve Bayes
  - Collaborative filtering
  - Singular value decomposition

Exploiting Sparsity in K-Means

Training set:

- number of examples: 12 million
- number of features: 500
- density: 10%

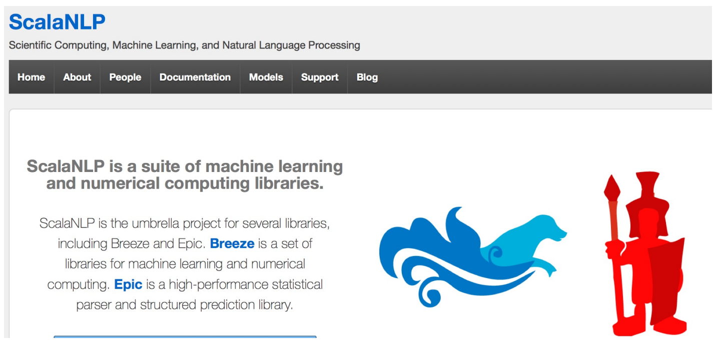
	dense	sparse
storage	47GB	7GB
time	240s	58s

Not only did we save 40GB of storage by switching to the sparse format, but we also received a 4x speedup.

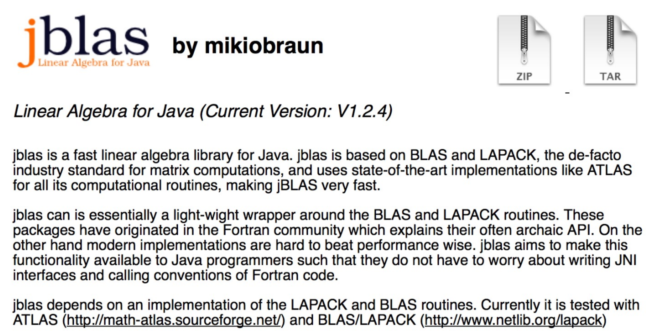
Xiangrui Meng, Databricks 2013

Main Data Types #1

- Spark supports local and distributed vectors and matrices
- Stored in one or more RDDs or DataFrames
- Linear algebra operations are provided by:
  - Breeze



  - jblas



Main Data Types #2

- Local vector:
  - Dense: numPy array or Python list
  - Sparse: MLlib's Sparse Vector

- Local matrix:
  - Support for dense matrices

- Distributed matrix:
  - Distributed in one or more data sets
  - Various formats to store large, distributed matrices:
    - RowMatrix
    - IndexedRowMatrix
    - CoordinateMatrix

Zurich University
of Applied Sciences

Example: Dense and Sparse Vectors

```python
import numpy as np
import scipy.sparse as sps
from pyspark.mllib.linalg import Vectors

# Use a NumPy array as a dense vector.
dv1 = np.array([1.0, 0.0, 3.0])
# Use a Python list as a dense vector.
dv2 = [1.0, 0.0, 3.0]
# Create a SparseVector.
sv1 = Vectors.sparse(3, [0, 2], [1.0, 3.0])
```

Number of elements

Indices for non-zero elements

Values

Note: Dense and sparse implementations support the same operations

Conclusions

- Spark ML enables scalable machine learning
- New algorithms are added (need to be "parallelized")
- Next big thing - Deep Learning Pipelines:
  - Use existing deep learning libraries such as TensorFlow or PyTorch with Spark

Deep Learning Guide

Databricks provides an environment that makes it easy to build, train, and deploy deep learning (DL) models at scale. Many deep learning libraries are available in Databricks Runtime ML, a machine learning runtime that provides a ready-to-go environment for machine learning and data science. For deep learning libraries not included in Databricks Runtime ML, you can either install libraries as a Databricks library or use init scripts to install libraries on clusters upon creation.

Graphics processing units (GPUs) can accelerate deep learning tasks. For information about creating GPU-enabled Databricks clusters, see GPU-enabled Clusters. Databricks Runtime includes pre-installed GPU hardware drivers and NVIDIA libraries such as CUDA.

https://docs.databricks.com/applications/deep-learning/index.html

Outlook: Next Week's Lecture

- For students who have NOT attended KI1/KI2:
  - Question & answering session on machine learning topics
- For students who have attended KI1/KI2:
  - Check out bonus material on deep learning in Moodle