

Lecture 1: Introduction to Spark and HPC

[Haiping Lu](#)

[COM6012: Scalable ML](#)

YouTube Playlist:

<https://www.youtube.com/c/HaipingLu/>



Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

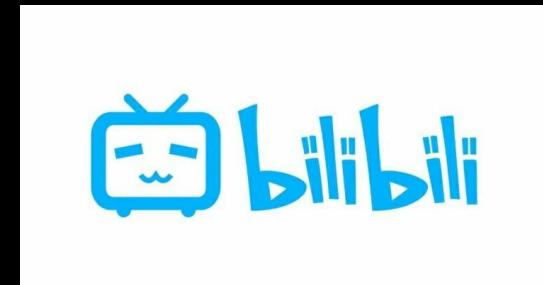
Where Does Big Data Come From?

- All happening **online**, e.g. tracking of:
 - Clicks
 - Billing events
 - Server requests
 - Transactions
 - Network messages
 - Faults
 - ...

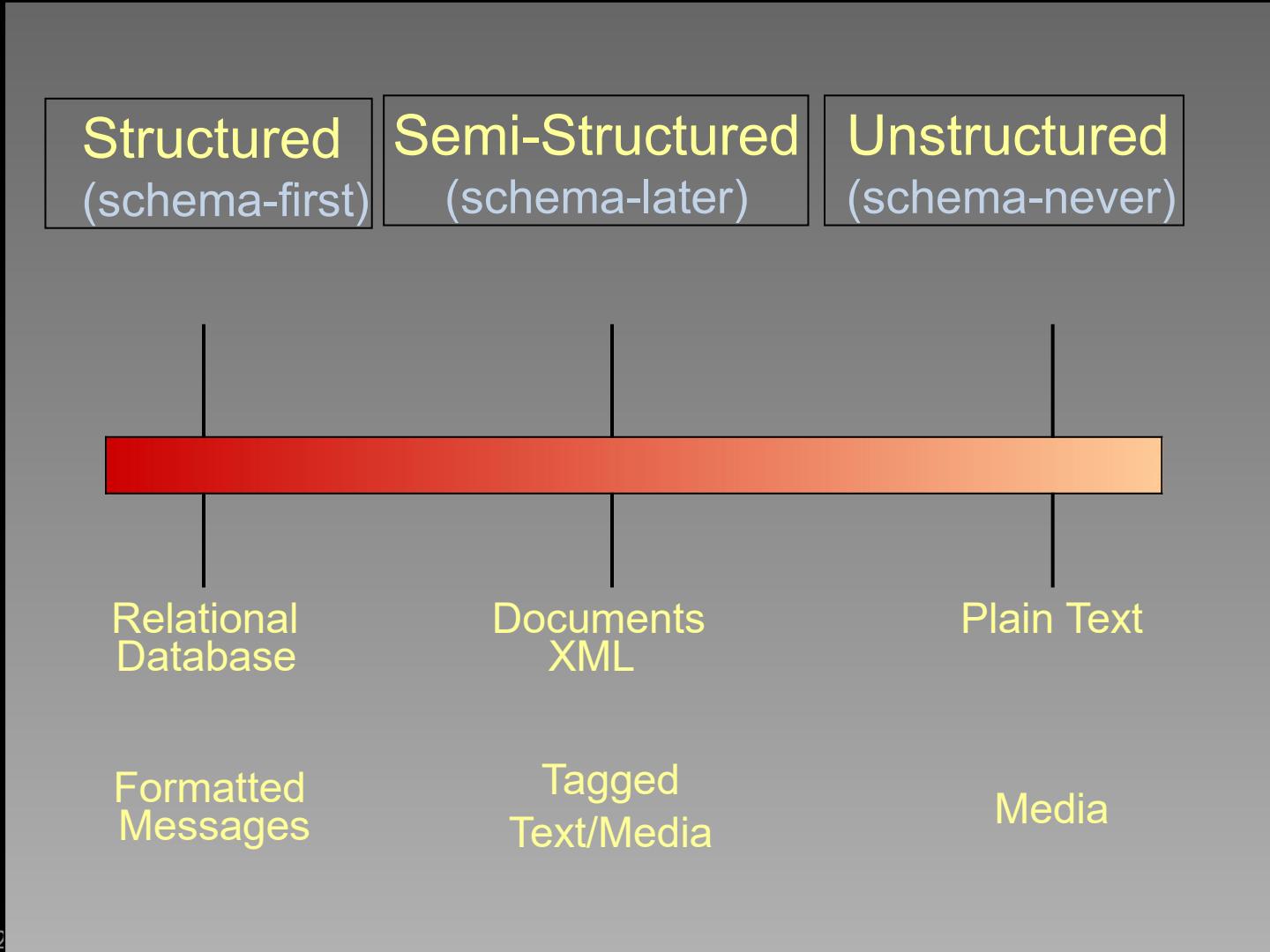


Where Does Big Data Come From?

- User generated content: web + mobile

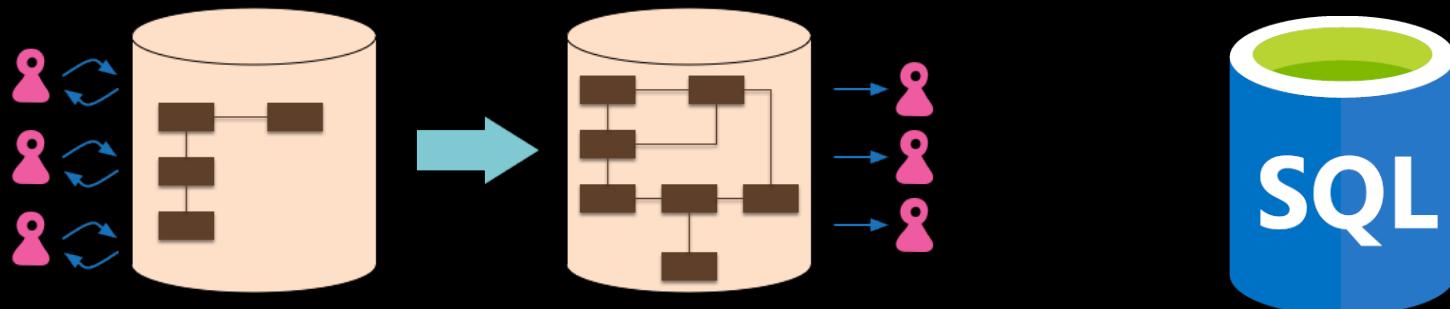


Data Structure Spectrum



Structured Data

- **Database:** relational data model → how a database is structured and used (hottest job 20 years ago)
- **Schema:** the organisation of data as a blueprint of how the database is constructed
 - The programmer **must statically specify** the schema
 - Decreasing ← consumer/media app, enterprise search
- **SQL:** Structured Query Language



Semi-Structured Data

- **Self-describing** rather than formal structures, tags/markers to separate semantic elements
- The column types → the **schema** for the data
 - Spark dynamically infers the schema while reading each row
 - Programmer statically specifies the schema
- Examples:



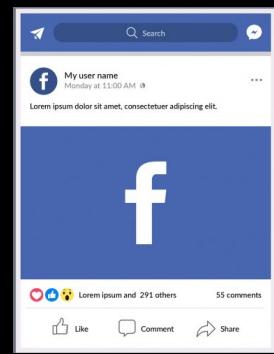
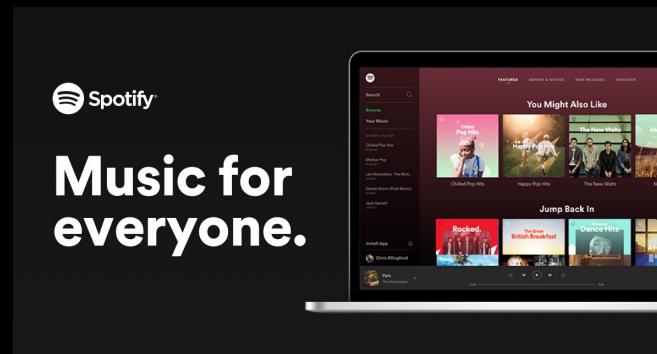
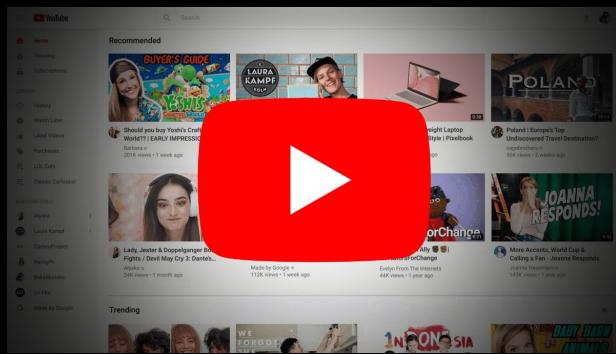
```
<?xml version="1.0"?>
<quiz>
  <qanda seq="1">
    <question>
      Who was the forty-second
      president of the U.S.A.?
    </question>
    <answer>
      William Jefferson Clinton
    </answer>
  </qanda>
  <!-- Note: We need to add
       more questions later.-->
</quiz>
```

XML



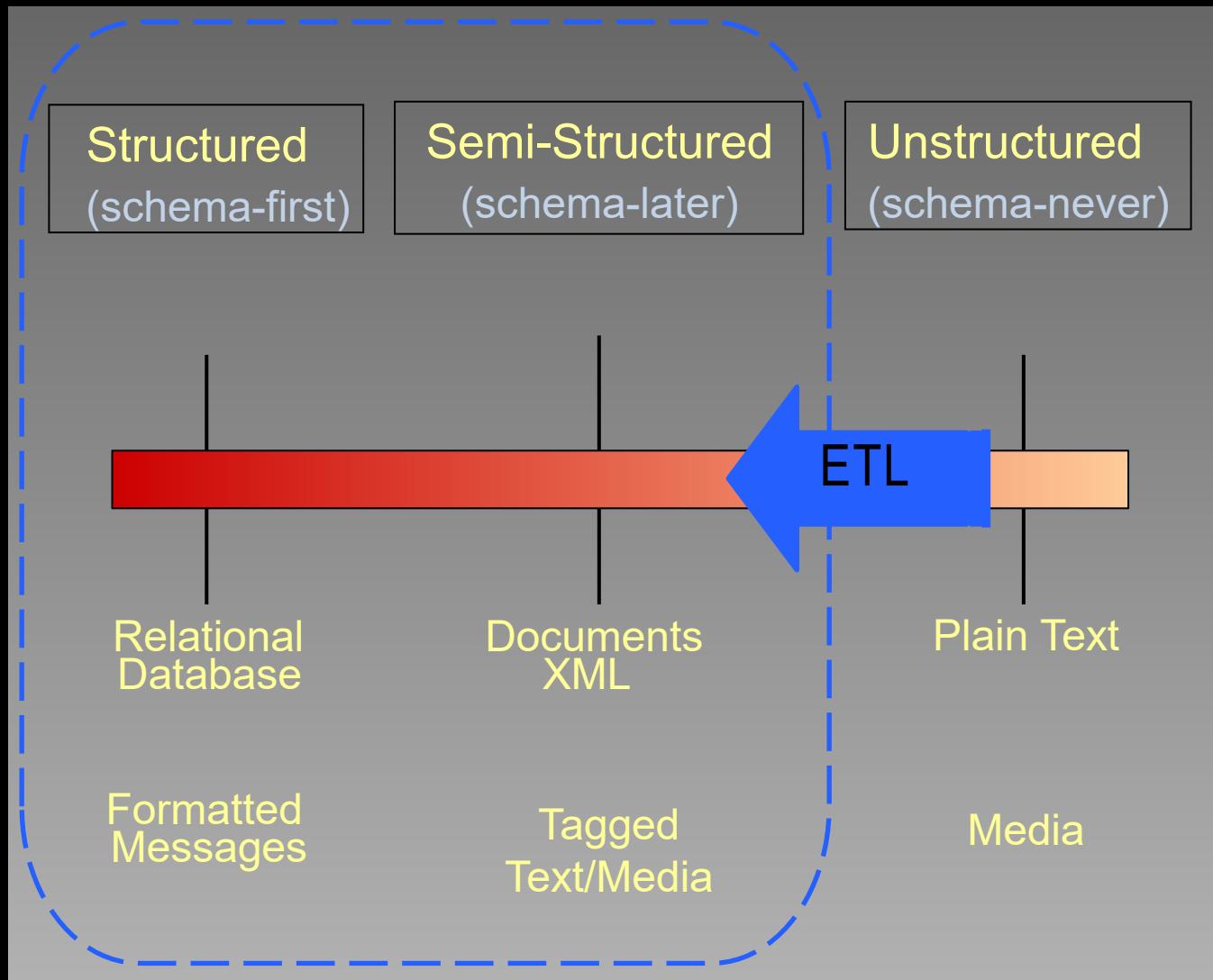
Unstructured Data

- Only one column with string or binary type
- Examples



- More than 70%–80% of all data in organisations
(Shilakes 1998)

Traverse the Data Structure Spectrum



- Impose structure on unstructured data
 - Extract
 - Transform
 - Load

Traditional Analysis Tools

- Unix shell commands (awk, grep, ...)

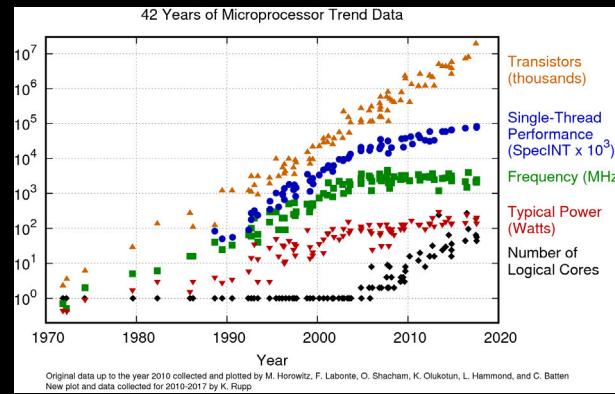
```
root@nginx:~# awk '{print $0}' file.txt
Item      Model      Country          Cost
1         BMW        Germany        $25000
2         Volvo       Sweden        $15000
3         Subaru      Japan         $2500
4         Ferrari    Italy        $2000000
5         SAAB        USA          $3000
```

```
vulphere@arifuretaarch:~|⇒ grep root /etc/passwd
root:x:0:0:root:/root:/bin/zsh
vulphere@arifuretaarch:~|⇒ grep -n root /etc/passwd
1:root:x:0:0:root:/root:/bin/zsh
vulphere@arifuretaarch:~|⇒ grep -c false /etc/passwd
3
vulphere@arifuretaarch:~|⇒ _
```

All run on a single machine!

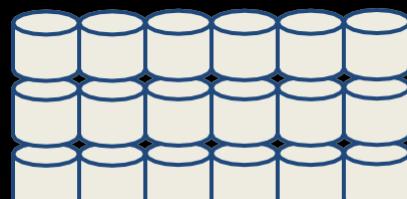
The Big Data Problem

- Data growing faster than computation speeds
- Growing data sources
 - Web, mobile, scientific, ...
- Storage getting cheaper
 - Size doubling every 18 months
- But, stalling CPU speeds and storage bottlenecks

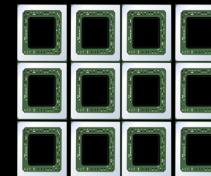


Solution for the Big Data Problem

- One machine can not process or *even store* all the data!
- Solution: distribute data over a **cluster** of machines



Lots of hard drives



... and CPUs



... and memory!

Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

Apache Spark

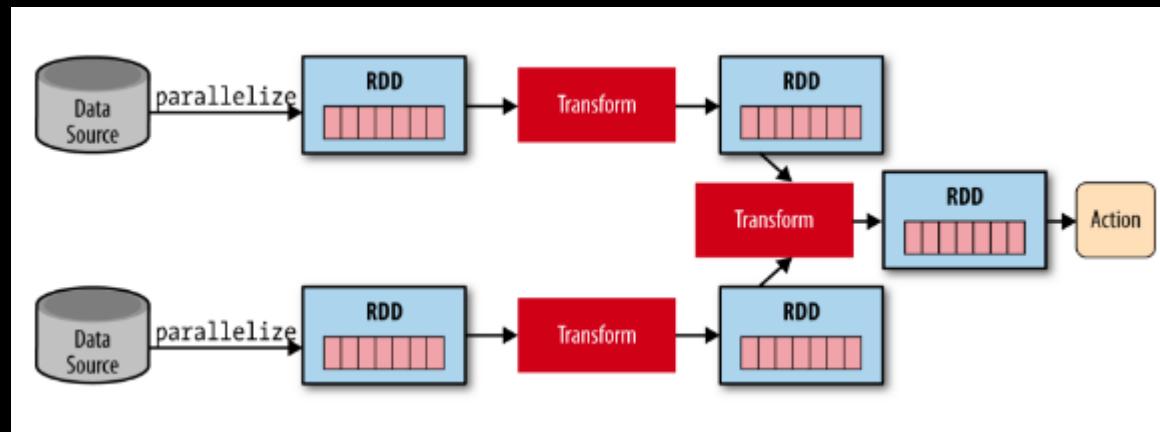
- Fast and general **cluster** computing system
- Interoperable with The logo for Hadoop MapReduce features a yellow elephant icon on the left, followed by the word "hadoop" in blue lowercase letters, and "Map Reduce" in yellow lowercase letters below it.
- Improves efficiency through:
 - In-memory computing primitives
 - General computation graphs
- Improves usability through:
 - Rich APIs in Scala, Java, **Python**
 - Interactive shell

→ Up to 100× faster
(2-10× on disk)

→ 2-5× less code

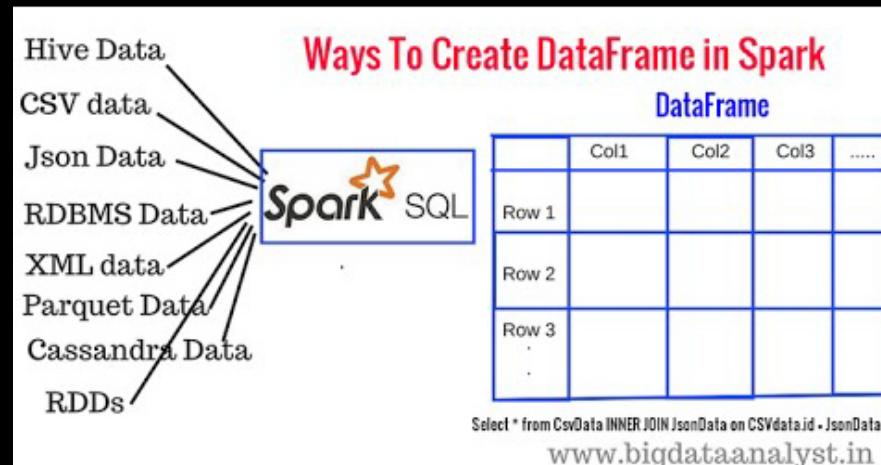
Spark Model

- Write programs in terms of **transformations** on **distributed** datasets
- Resilient Distributed Datasets (RDDs)
 - **Collections** of objects that can be stored in memory or disk across a cluster
 - **Parallel** functional transformations (map, filter, ...)
 - Automatically rebuilt on **failure**



Spark for Data Science

- DataFrames
 - Structured data (**SQL**)
 - Familiar API based on R/Python Pandas
 - Distributed, optimised implementation
- Machine learning pipelines
 - Simple construction and tuning of **ML workflows**



Spark Computing Framework

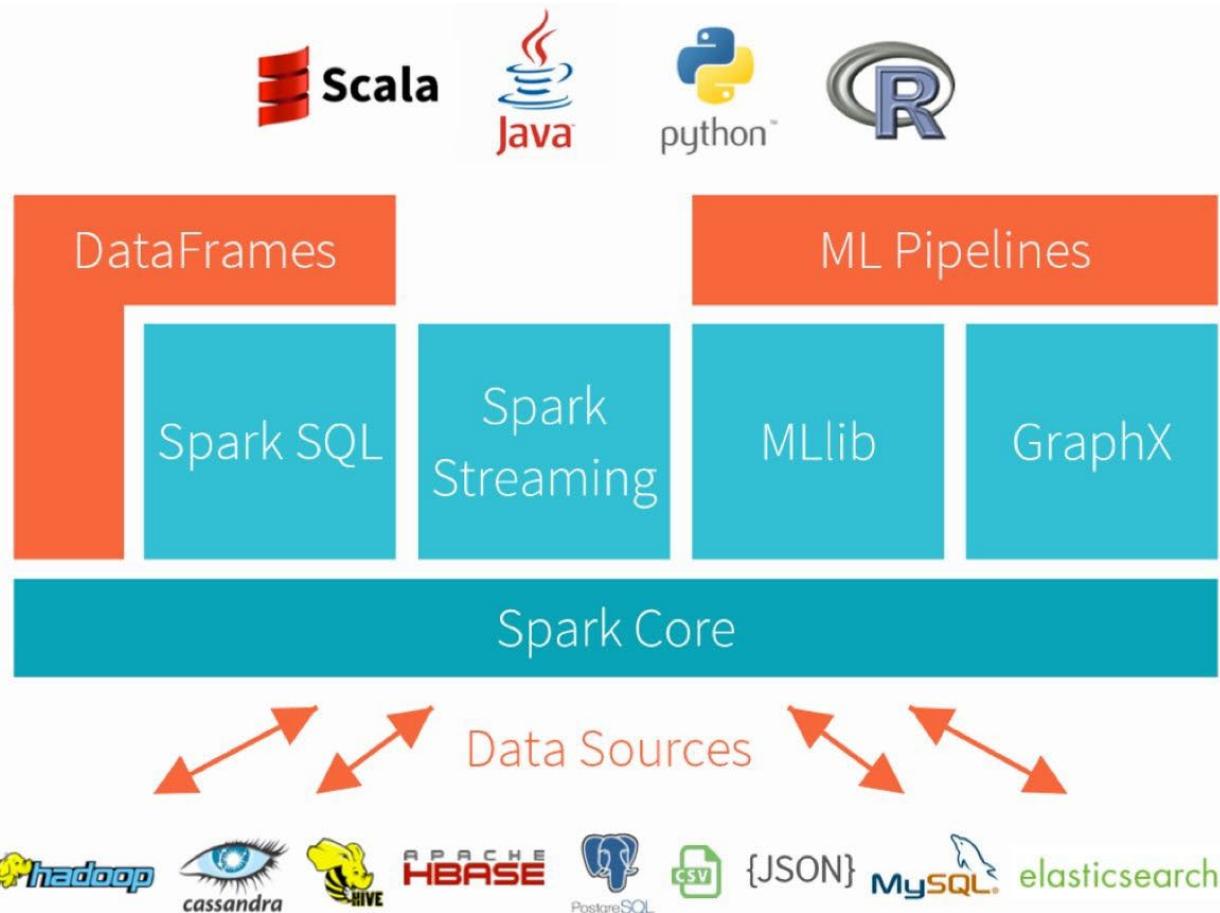
- Programming abstraction and parallel runtime to hide complexities of **fault-tolerance** and **slow machines**

“Here’s an operation, run it on all of the data”

JUST DO IT.

- I don’t care where it runs (you schedule that)
- In fact, feel free to run it twice on different nodes (e.g. when it fails)

Apache Spark Ecosystem



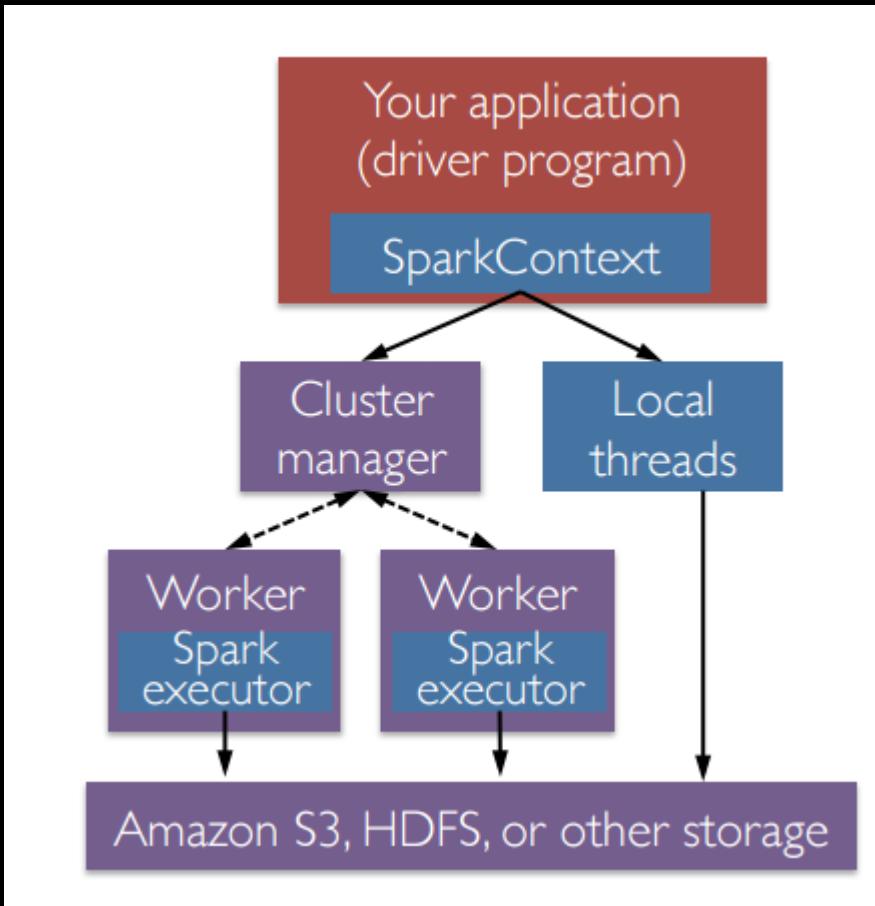
Total contributors: 150 → 500

Lines of code: 190K → 370K

500+ active production deployments

<https://i.pinimg.com/originals/e7/f3/2d/e7f32d041846a5938a09e192bdf3885d.jpg>

Spark Components



- A Spark program first creates a **SparkSession** object as the driver (including **SparkContext**)
 - Tells Spark how/where to access a cluster
 - Connect to cluster managers
- Cluster managers
 - Allocate resources across applications
- **Spark executor (worker):**
 - Run computations
 - Access data storage

SparkSession and SparkContext

- SparkSession
 - Entry point for DataFrame API, create **DataFrames**
 - PySpark shell automatically create SparkSession as **spark**
 - Programs: must create a new SparkSession first (see lab)
- SparkContext
 - Entry point for Spark functionality, create **RDDs**
 - Connect to a Spark cluster
 - Associated with a SparkSession
 - PySpark shell automatically create SparkContext as **sc**
 - Programs: **sc = spark.sparkContext**

The '*Master*' Parameter for a SparkSession

- Determines cluster type and size

Master Parameter	Description
local	run Spark locally with one worker thread (no parallelism)
local[K]	run Spark locally with K worker threads (ideally set to number of cores)
spark://HOST:PORT	connect to a Spark standalone cluster; PORT depends on config (7077 by default)
mesos://HOST:PORT	connect to a Mesos cluster; PORT depends on config (5050 by default)

Week 1 Contents / Objectives

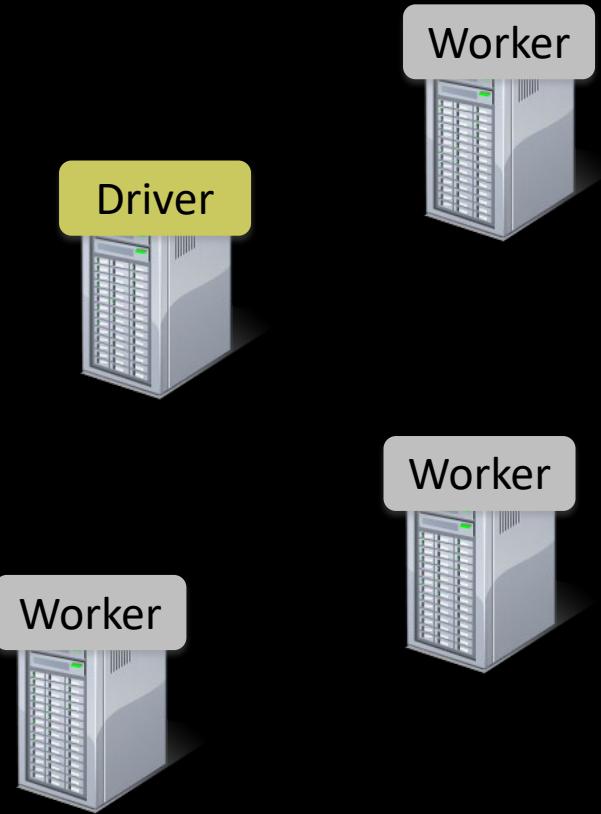
- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

Spark Example: Log Mining (w/t RDD)

Load error messages from a log into memory, then interactively search for various patterns

Spark Example: Log Mining

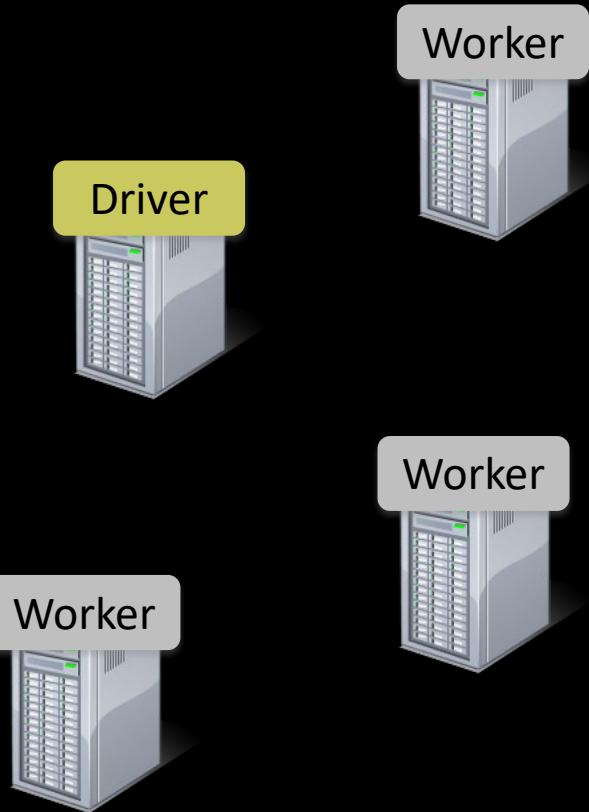
Load error messages from a log into memory, then interactively search for various patterns



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

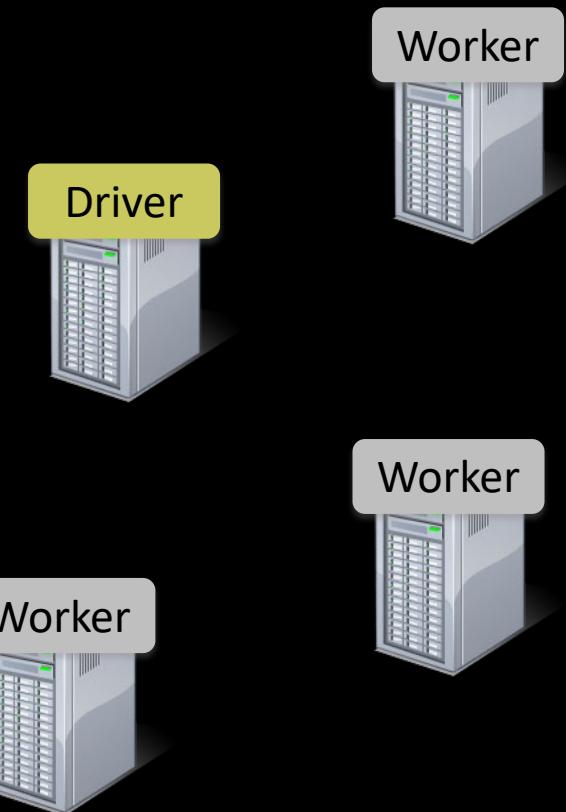
```
lines = spark.textFile("hdfs://...")
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

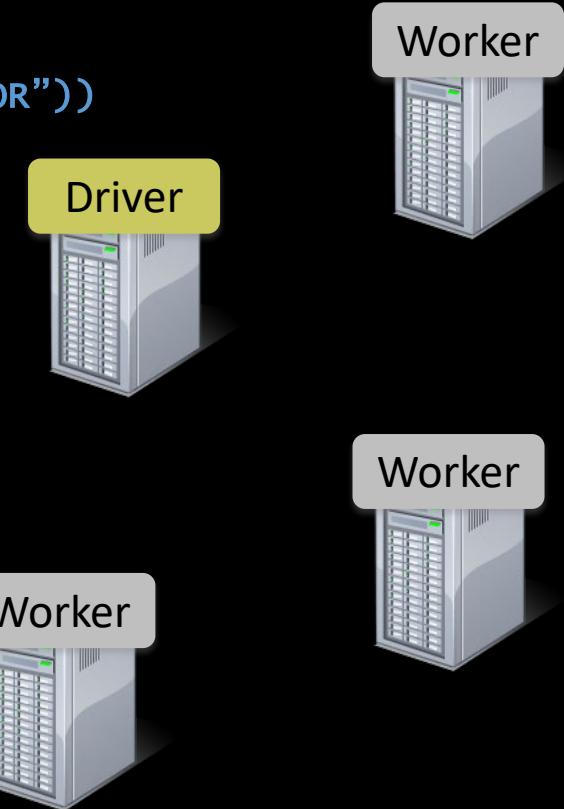
Base RDD
lines = spark.textFile("hdfs://...")



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

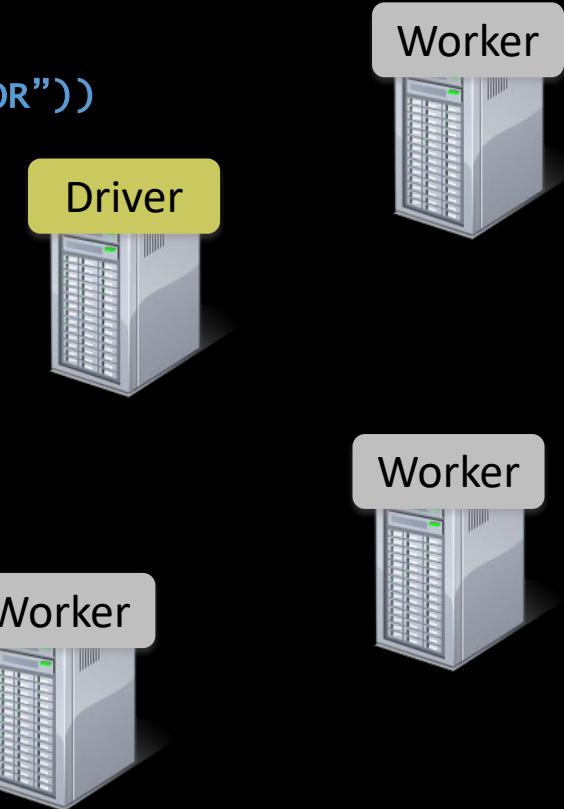
```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
Transformed RDD  
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

Driver



Worker



Worker

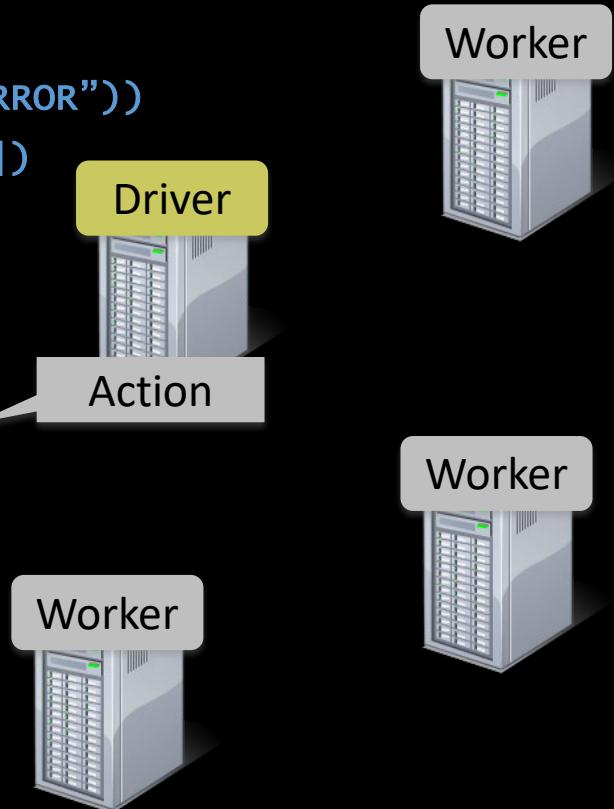


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

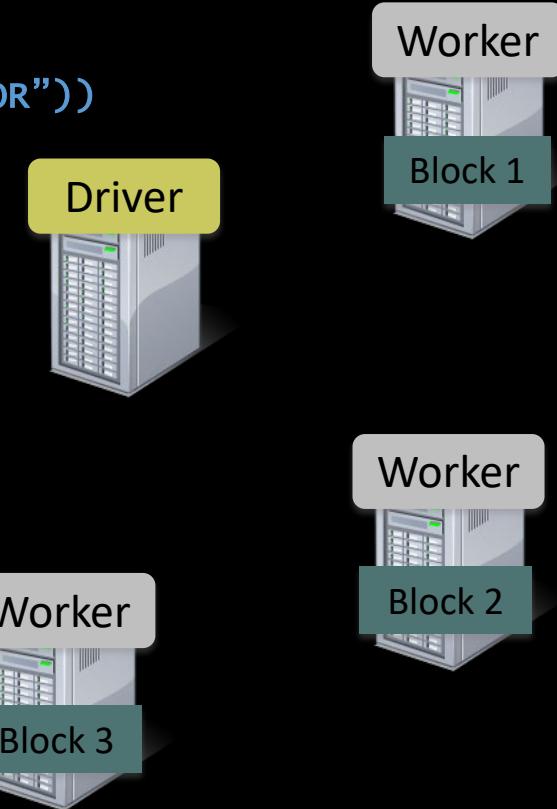


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

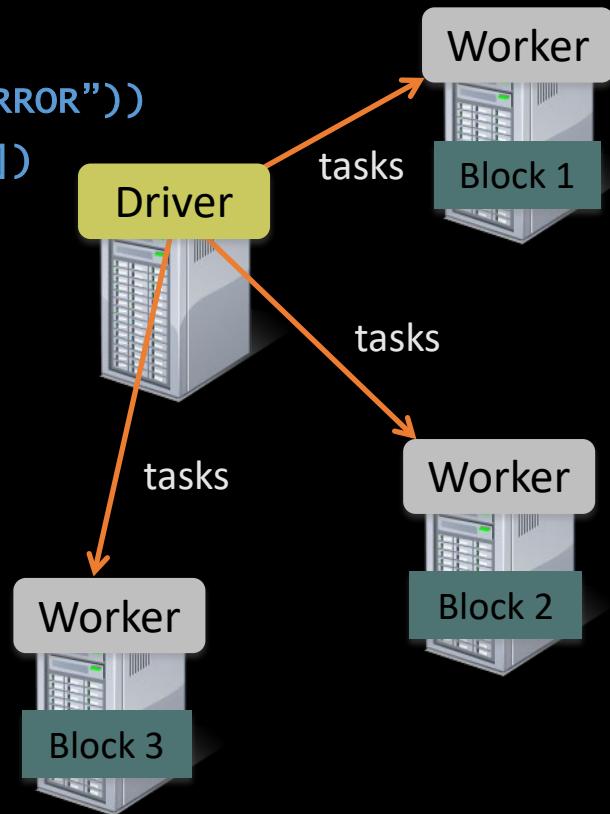


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

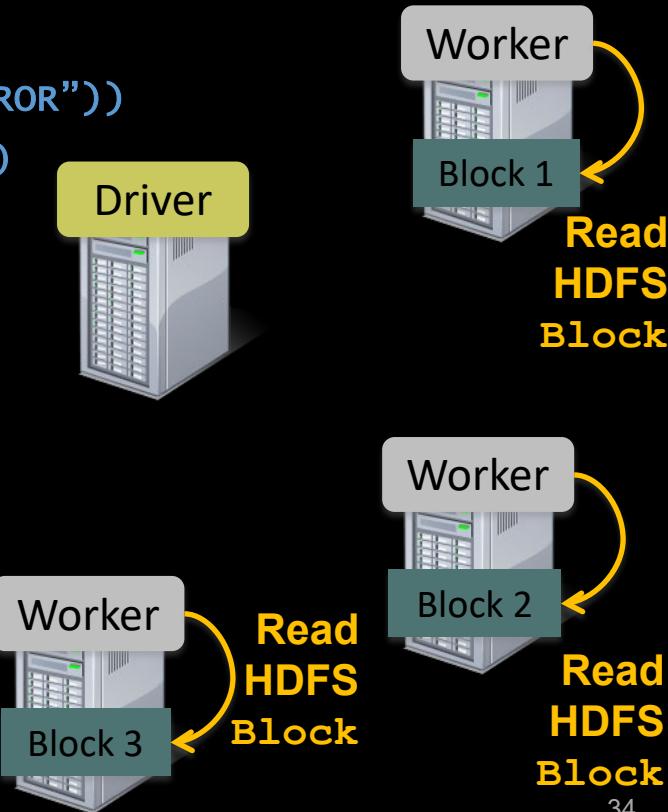


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

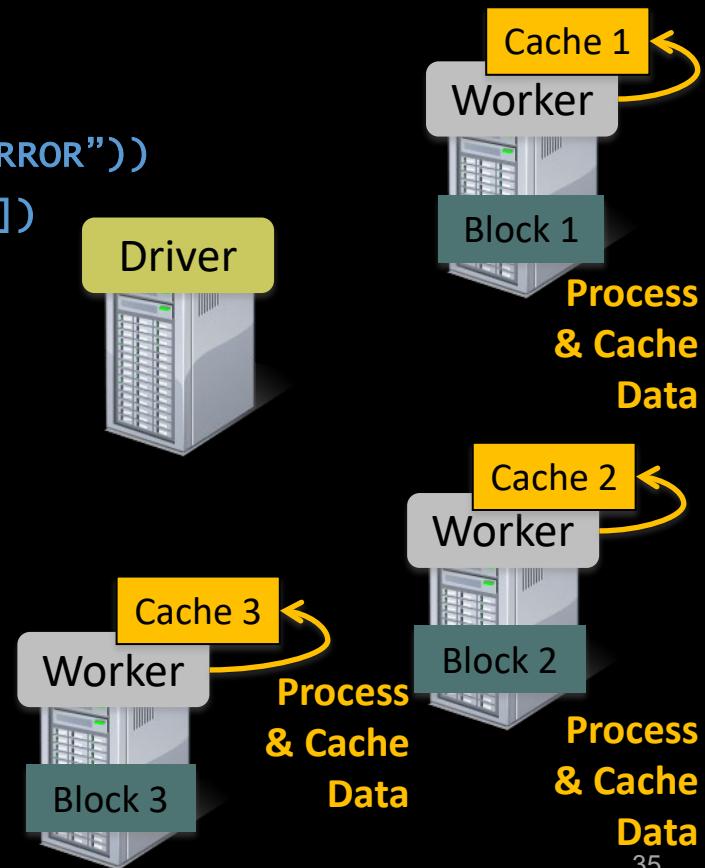
```
messages.filter(lambda s: "mysql" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()  
  
messages.filter(lambda s: "mysql" in s).count()
```

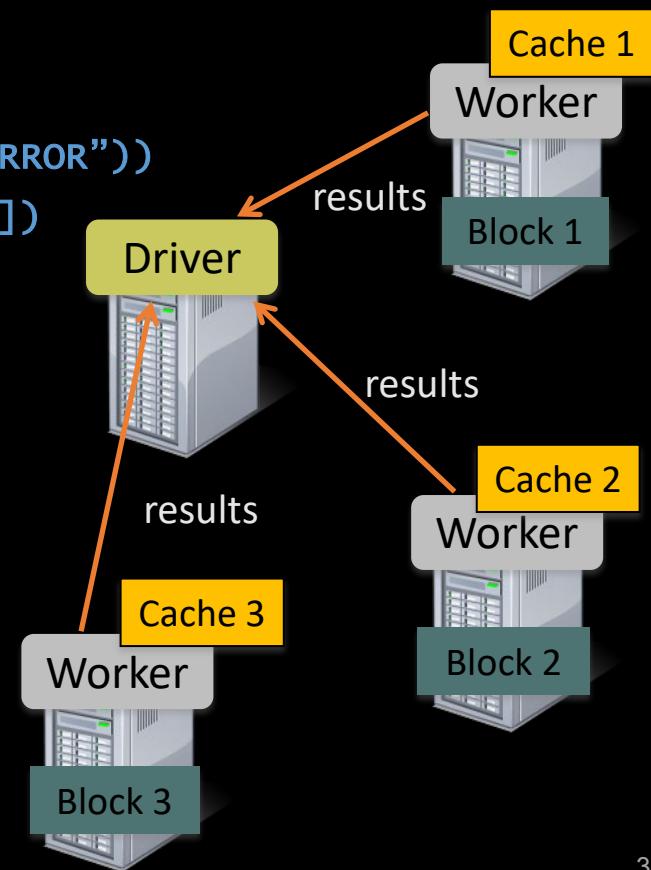


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()
```

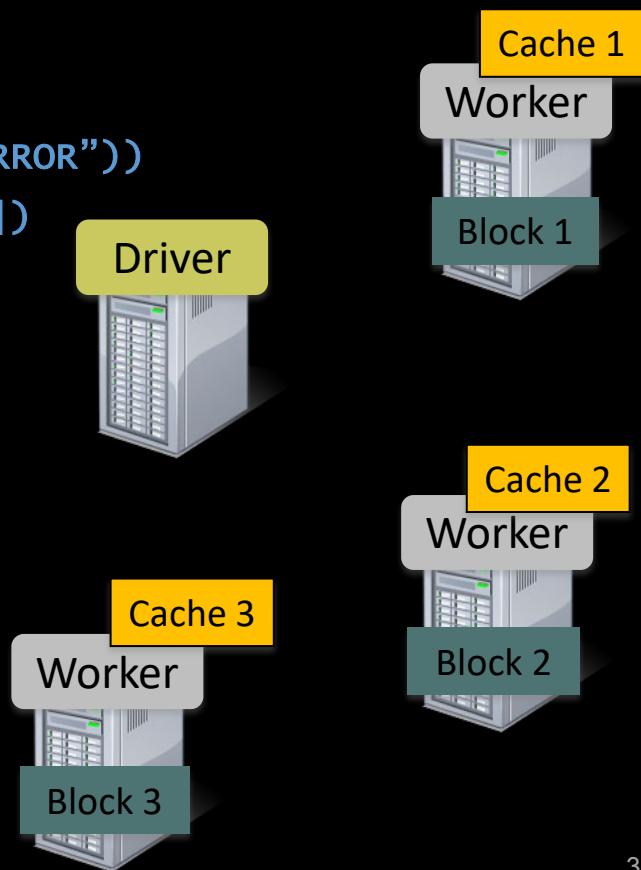


Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

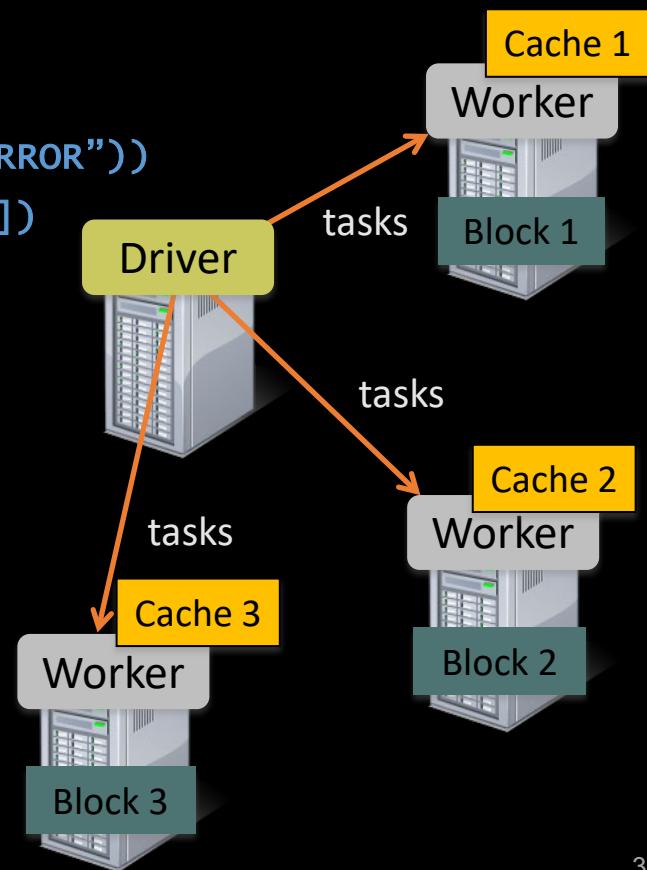
```
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

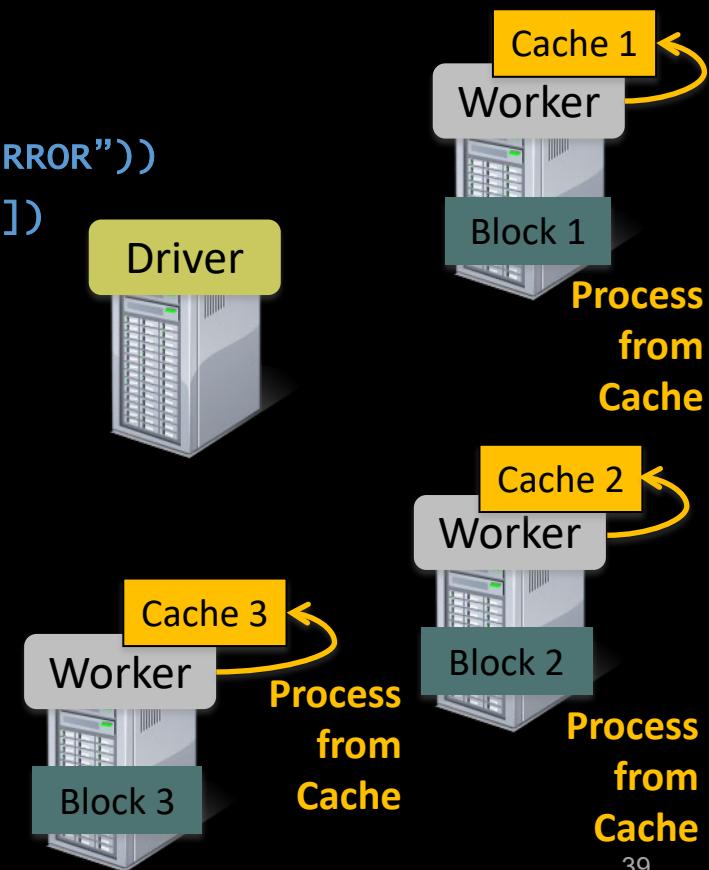
```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()  
  
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

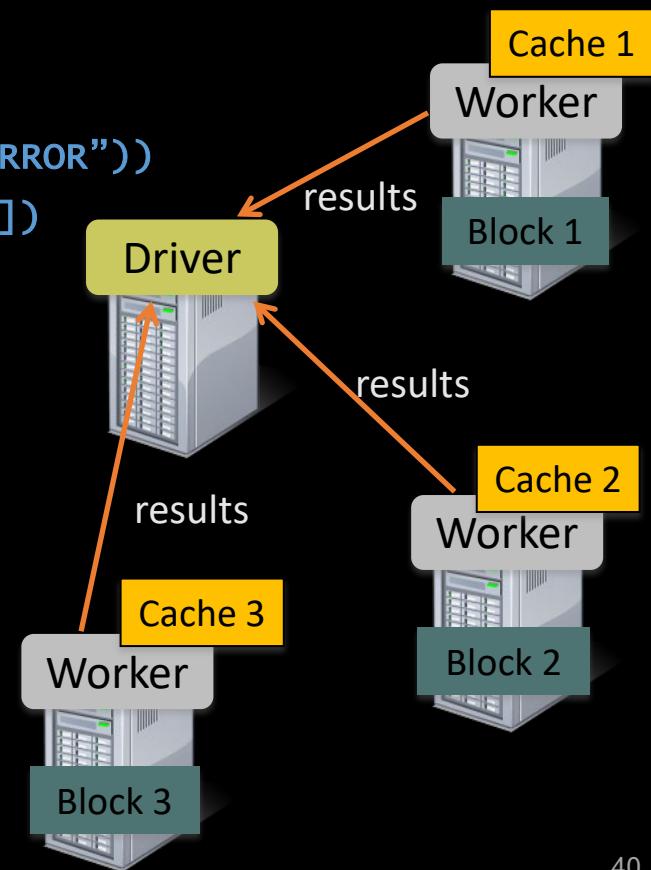
```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()  
  
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()  
  
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()
```



Spark Example: Log Mining

Load error messages from a log into memory, then interactively search for various patterns

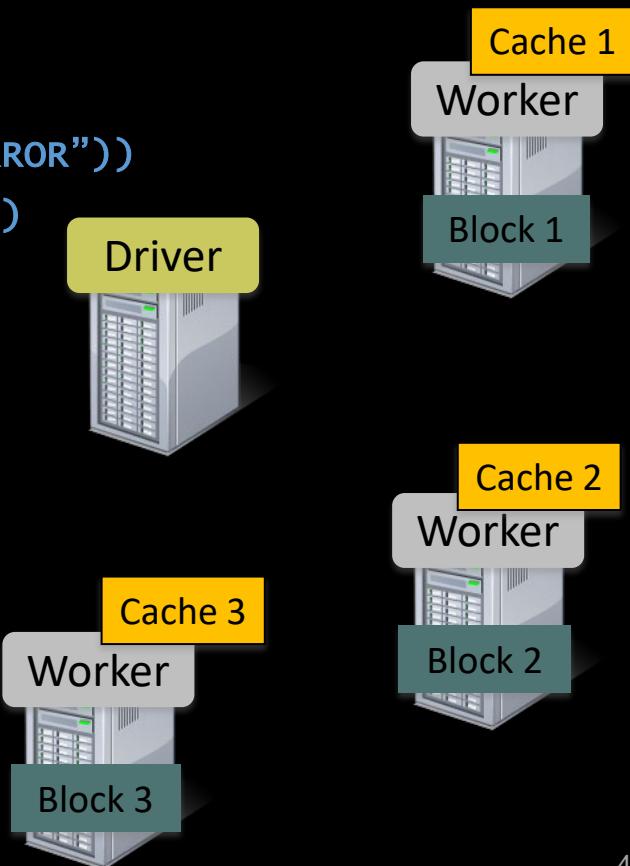
```
lines = spark.textFile("hdfs://...")  
errors = lines.filter(lambda s: s.startswith("ERROR"))  
messages = errors.map(lambda s: s.split("\t")[2])  
messages.cache()
```

```
messages.filter(lambda s: "mysql" in s).count()  
messages.filter(lambda s: "php" in s).count()
```

Cache your data → Faster results

Full-text search of Wikipedia

- 60GB on 20 EC2 machines
- 0.5 sec from mem vs. 20s for on-disk



Week 1 Contents / Objectives

- The Big Data Problem: Why Spark?
- What is Spark?: The Essentials
- An Example of Spark: Log Mining
- How to Use Spark: PySpark, HPC, Resources

Spark Program Lifecycle

- Create DataFrames from external data or [createDataFrame](#) from a collection in a driver program
- Lazily transform them into new DataFrames
- [cache\(\)](#) some DataFrames for reuse
- Perform [actions](#) to execute parallel computation and produce results

Use Spark Transformations and Actions wherever possible: Search [DataFrame reference API](#)

PySpark 3.2.1

- Need: Java, Python, Spark
- See lab 1 on how to install on HPC
- To install on Windows (optional)
 - Lab 1 instructions: Install Java JRE, Python, Spark
 - Or pip install pyspark==3.2.1
- To install on Linux/Mac (optional): see lab references



ShARC HPC @ Sheffield

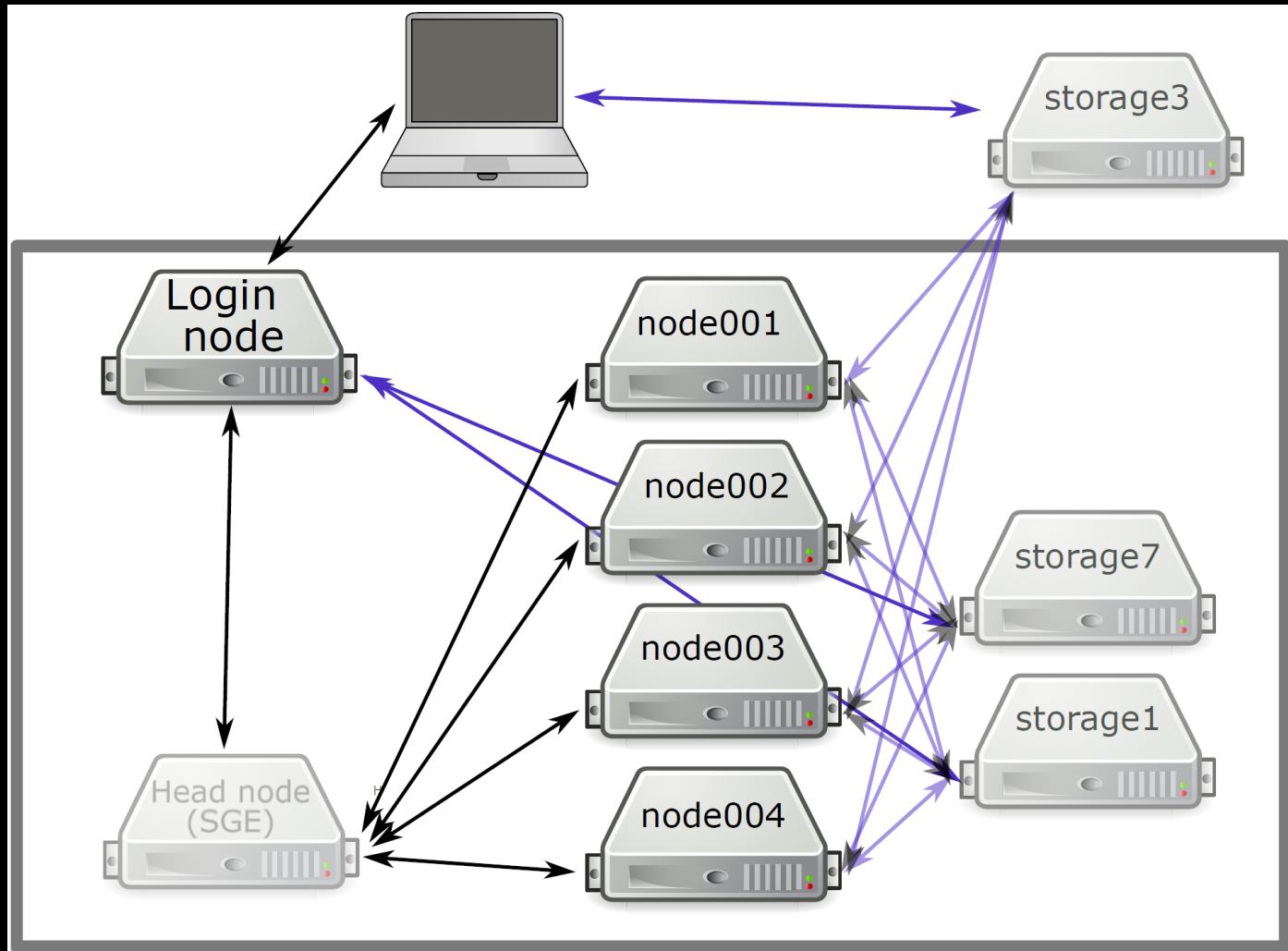
- ShARC: Sheffield Advanced Research Computer
- VPN: a **MUST** unless you are on a **campus network**
- Training: HPC Driving License test
- SSH access via **sharc.sheffield.ac.uk**
 - Windows: MobaXTerm
 - Linux/MAC OS: terminal (command line)
- Help: it-servicedesk@sheffield.ac.uk



Haiping Lu - University of Sheffield



HPC Cluster Structure



Storage

Location	Shared	Quota	Back ups	Speed	Suitable for?
/home/\$USER	Y	10GB	Y	>	Personal data
/data/\$USER	Y	100GB	Y	>	Personal data
/fastdata/\$USER	Y	-	N	>>>	Temporary big files
/scratch	N	-	N	>>>	Temporary small files



Interactive Session

```
pyspark
```

```
.....
```

```
Welcome to
```

```
 _/ \_ / \_ _ _ _ / / _  
 _\ \V _ \V _ ` \ / _ / ' _ /  
 /_ / . _ / \_, _/ / / _ / \ \_ / \ version 3.0.1  
 /_ /
```

```
Using Python version 3.6.2 (default, Jul 20 2017 13:51:32)
```

```
SparkSession available as 'spark'.
```

```
>>>
```

Batch Session – Shell Script xx.sh

Create a file `Lab1_SubmitBatch.sh`

```
#!/bin/bash
#$ -l h_rt=6:00:00 #time needed
#$ -pe smp 2 #number of cores
#$ -l rmem=8G #number of memery
#$ -o ../Output/COM6012_Lab1.txt #This is where your output and errors are logged.
#$ -j y # normal and error outputs into a single file (the file above)
#$ -M youremail@shef.ac.uk #Notify you by email, remove this line if you don't like
#$ -m ea #Email you when it finished or aborted
#$ -cwd # Run job from current directory

module load apps/java/jdk1.8.0_102/binary

module load apps/python/conda

source activate myspark

spark-submit ../Code/LogMiningBig.py # .. is a relative path, meaning one level up
```

Batch Session: Submit & Relax

- **qsub** your job (can run at the login node): see Lab 1
- Then?
 - Close the terminal and leave
 - Wait for pre-set email notification
 - Check status: **qstat**
 - Cancel/amend job: **qdel**
- How much resources to request
 1. Run **short** test jobs
 2. View resource utilisation
 3. Extrapolate
 4. Submit larger jobs



Spark Resources

- [Apache Spark Documentation](#)
- [PySpark tutorial](#)
- [Spark videos on YouTube](#)
- [Open source code](#)
- Suggested reading in labs

Suggested reading:

- Spark Overview
- Spark Quick Start (Choose **Python** rather than the default *Scala*)
- Chapters 2 to 4 of [**PySpark tutorial**](#) (several sections in Chapter 3 can be safely skipped)
- Reference: [**PySpark documentation**](#)
- Reference: [**PySpark source code**](#)

Acknowledgements

- Some slides (sec. 1) are modified from the [“Introduction to Apache Spark”](#) course by Prof. A. D. Joseph, University of California, Berkeley.
- This module benefits from many open resources. See the acknowledgement on our [GitHub page](#).
- There are many other resources that I have consulted but somehow lost track of the origins.





<https://cdn.eventfinda.co.nz/uploads/events/transformed/1330836-590621-34.png>

Lecture 2: Spark RDD, DataFrame, ML Pipelines, and Parallelization

[COM6012: Scalable ML by Haiping Lu](#)

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/>

Week 2 Contents / Objectives

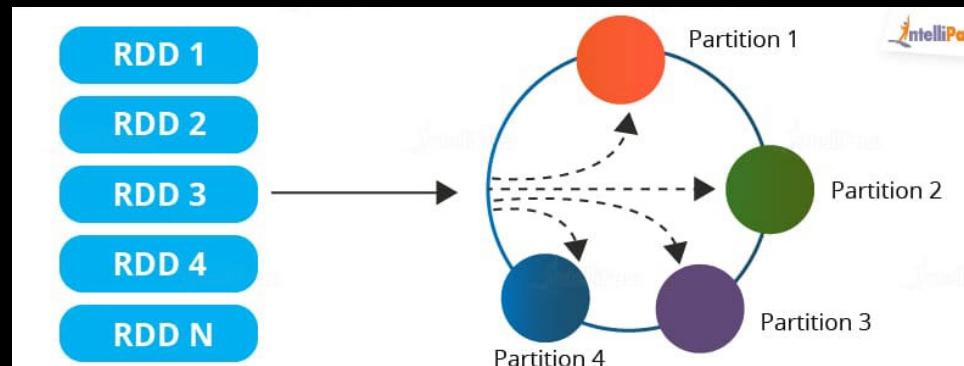
- Resilient Distributed Datasets
- DataFrames and Datasets
- Machine Learning Pipelines
- Execution Parallelization

Week 2 Contents / Objectives

- Resilient Distributed Datasets
- DataFrames and Datasets
- Machine Learning Pipelines
- Execution Parallelization

RDD

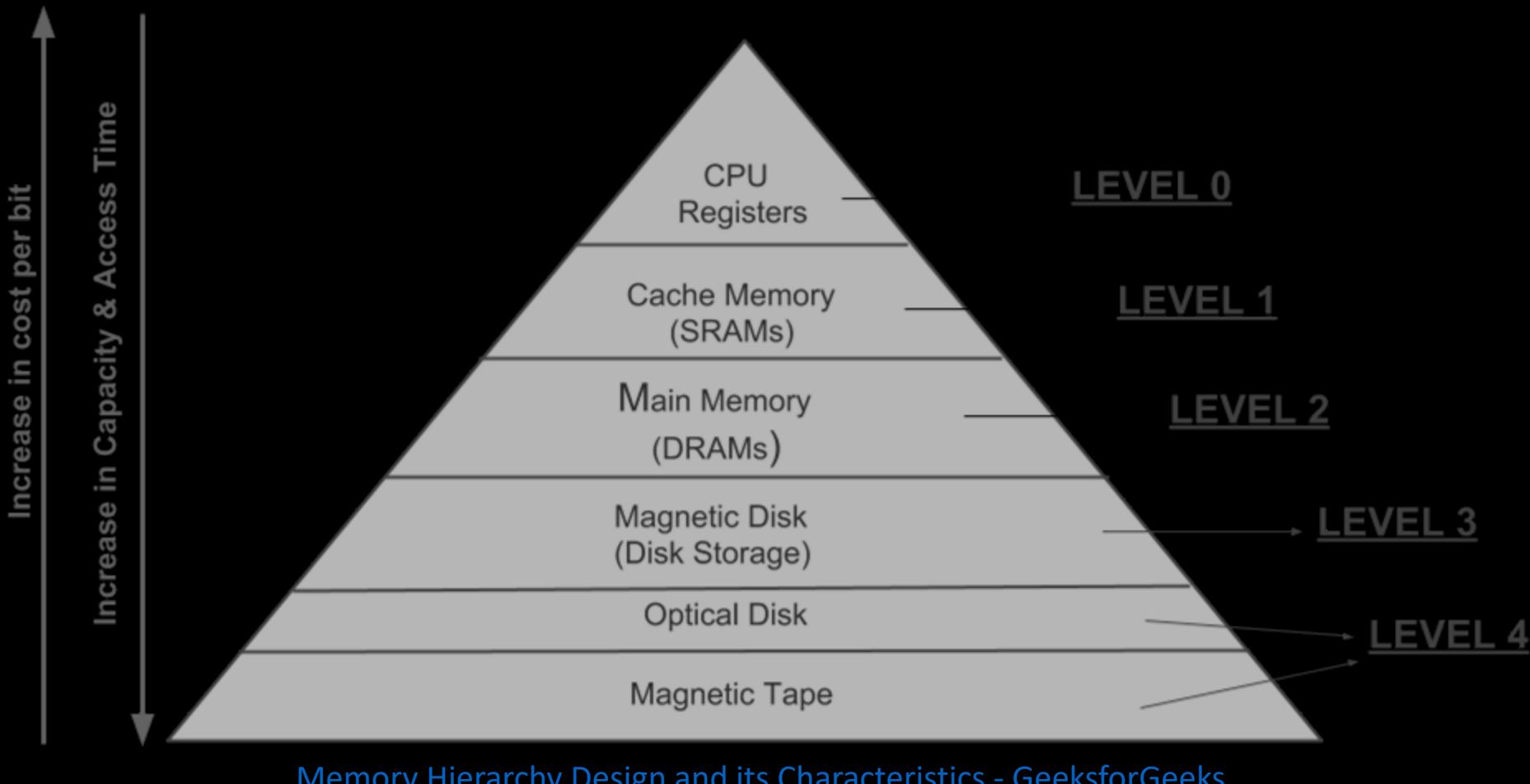
- Resilient Distributed Datasets
 - A **distributed** memory abstraction enabling **in-memory** computations on large **clusters** in a **fault-tolerant** manner
 - The **primary** data abstraction in Spark enabling operations on collection of elements in parallel
- **R**: recompute missing partitions due to node failures
- **D**: data **distributed** on multiple nodes in a cluster
- **D**: a collection of **partitioned** elements (**datasets**)



RDD Traits

- **In-Memory**: data inside RDD is stored in memory as much (size) and long (time) as possible
- **Immutable (read-only)**: no change after creation, only transformed using transformations to new RDDs
- **Lazily evaluated**: RDD data not available/transformed until an action is executed that triggers the execution
- **Parallel**: process data in parallel
- **Partitioned**: the data in a RDD is partitioned and then distributed across nodes in a cluster
- **Cacheable**: hold all the data in a persistent "storage" like memory (the most preferred) or disk (the least preferred)

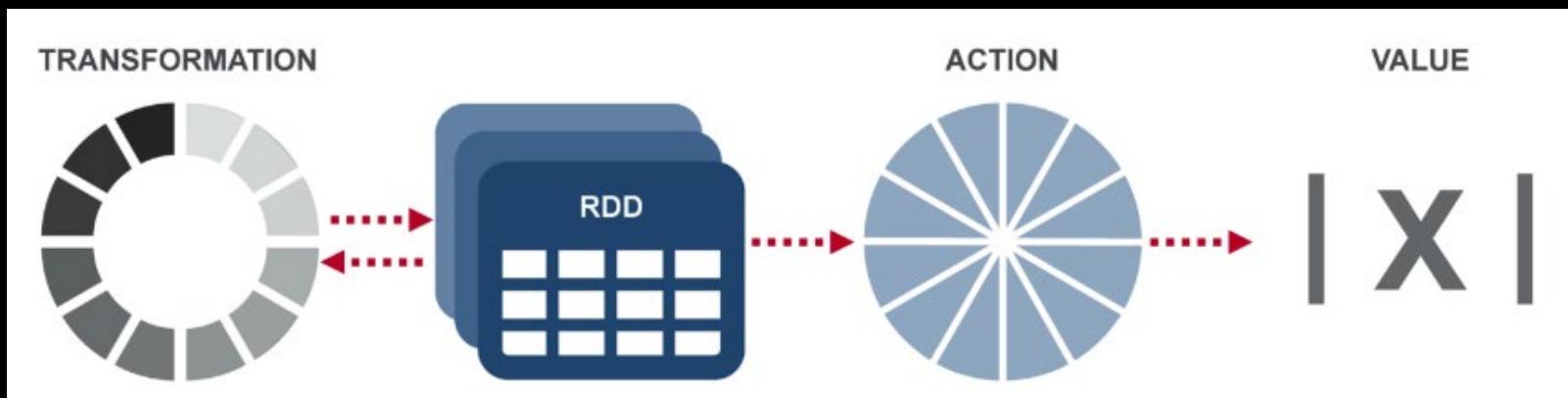
Computer Memory Hierarchy



[Memory Hierarchy Design and its Characteristics - GeeksforGeeks](#)

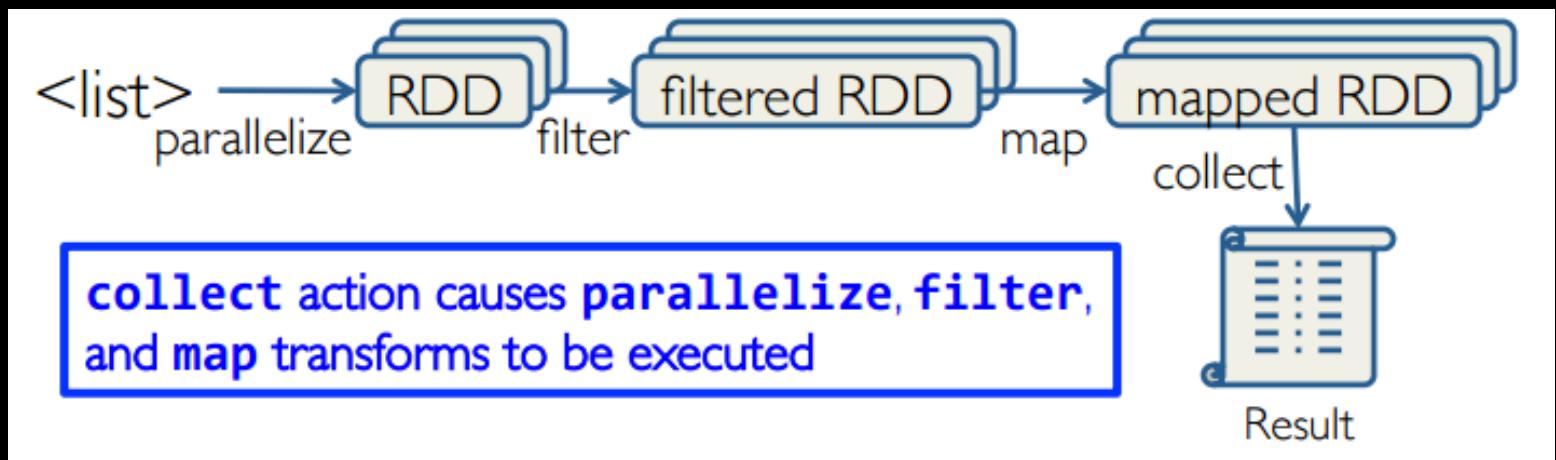
RDD Operations

- **Transformation:** takes an RDD and returns a new RDD but nothing gets evaluated / computed
- **Action:** all the data processing queries are **computed** (evaluated) and the result value is returned



RDD Workflow

- Create an RDD from a data source, e.g. RDD or file
- Apply transformations to an RDD, e.g., map, filter
- Apply actions to an RDD, e.g., collect, count
- Users to control 1) persistence, 2) partitioning



Creating RDDs

- Parallelize existing Python collections (lists)
- Transform existing RDDs
- Create from (HDFS, text, Amazon S3) files
- sc APIs: `sc.parallelize`, `sc.hadoopFile`, `sc.textFile`

Parallelized
Collections

From RDDs

External
Data

Spark Transformations

- Create new datasets from an existing one
- Lazy evaluation: just **remember** transformations applied to the base dataset (results not computed)
 - Spark optimises the required calculations
 - Spark recovers from failures

Transformation	Meaning
<code>map(func)</code>	Return a new distributed dataset formed by passing each element of the source through a function <i>func</i> .
<code>filter(func)</code>	Return a new dataset formed by selecting those elements of the source on which <i>func</i> returns true.
<code>flatMap(func)</code>	Similar to map, but each input item can be mapped to 0 or more output items (so <i>func</i> should return a Seq rather than a single item).
<code>mapPartitions(func)</code>	Similar to map, but runs separately on each partition (block) of the RDD, so <i>func</i> must be of type <code>Iterator<T> => Iterator<U></code> when running on an RDD of type T.

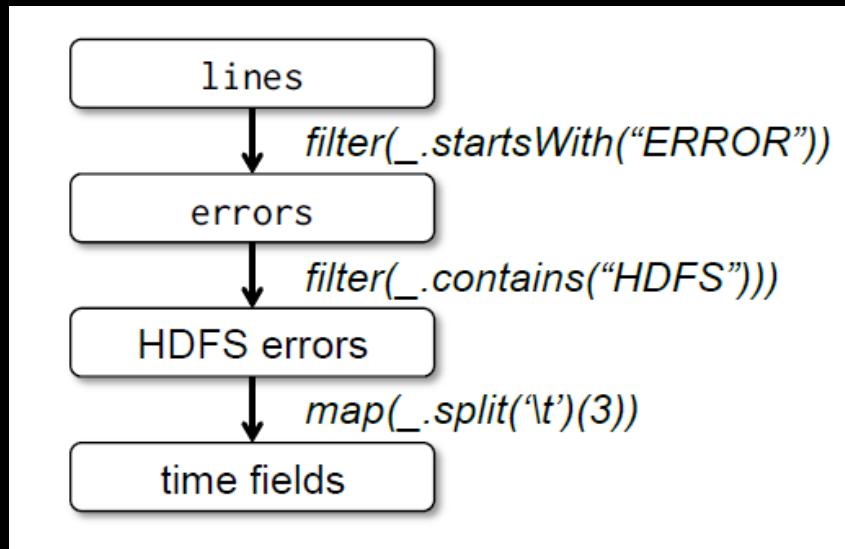
Spark Actions

- Cause Spark to execute recipe to transform source
- Mechanism for getting results out of Spark

Action	Meaning
<code>reduce(func)</code>	Aggregate the elements of the dataset using a function <i>func</i> (which takes two arguments and returns one). The function should be commutative and associative so that it can be computed correctly in parallel.
<code>collect()</code>	Return all the elements of the dataset as an array at the driver program. This is usually useful after a filter or other operation that returns a sufficiently small subset of the data.
<code>count()</code>	Return the number of elements in the dataset.
<code>first()</code>	Return the first element of the dataset (similar to <code>take(1)</code>).
<code>take(n)</code>	Return an array with the first <i>n</i> elements of the dataset.
<code>takeSample(withReplacement, num, [seed])</code>	Return an array with a random sample of <i>num</i> elements of the dataset, with or without replacement, optionally pre-specifying a random number generator seed.
<code>takeOrdered(n, [ordering])</code>	Return the first <i>n</i> elements of the RDD using either their natural order or a custom comparator.

Example from the Spark Paper (2012)

- Web service is experiencing errors. Operators want to search terabytes of logs in the Hadoop file system to find the cause.



Lineage Graph

//base RDD

Code in Scala

```
val lines = sc.textFile("hdfs://...")
```

//Transformed RDD

```
val errors = lines.filter(_.startsWith("Error"))
```

```
errors.persist() //or .cache()
```

```
errors.count()
```

```
errors.filter(_.contains("HDFS"))
```

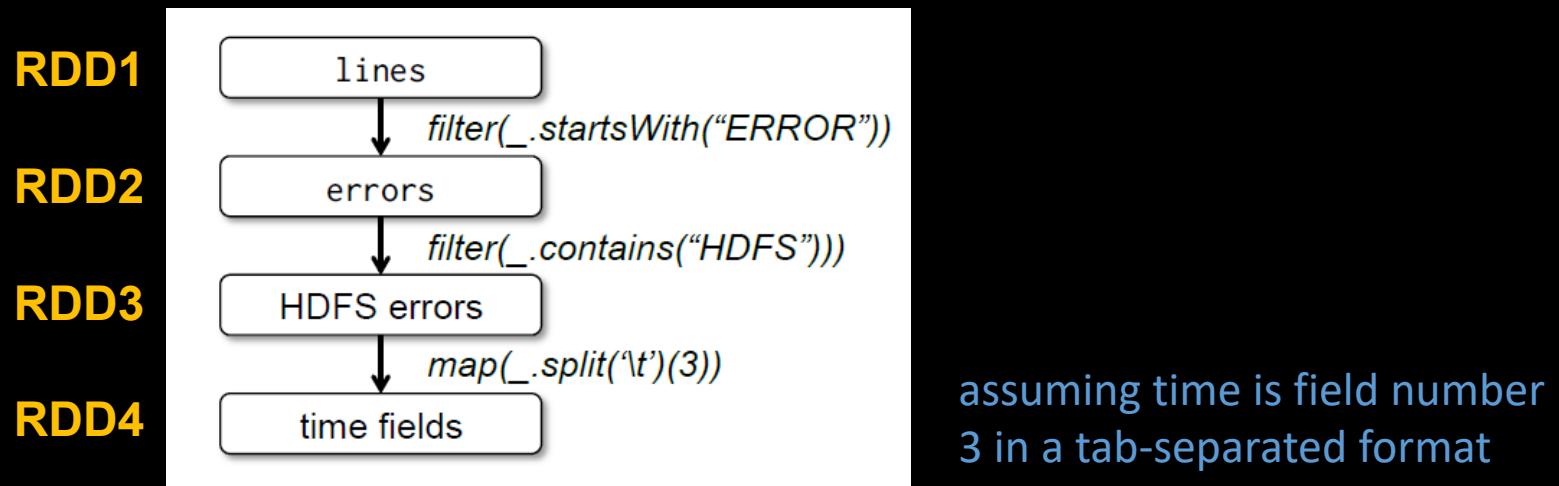
```
.map(_.split('\\t')(3))
```

```
.collect()
```

- Line1: create RDD from an HDFS file (but NOT loaded in memory)
- Line3: ask for errors to persist in memory (when loaded)

Lineage Graph → Fault-Tolerance

- RDDs keep track of **lineage** → how it was derived from to compute its partitions from data in **stable** storage
- A partition of errors is lost → rebuild it by applying a filter on **only** the corresponding partition of lines → partitions can be recomputed in parallel on different nodes without rolling back the **whole** program

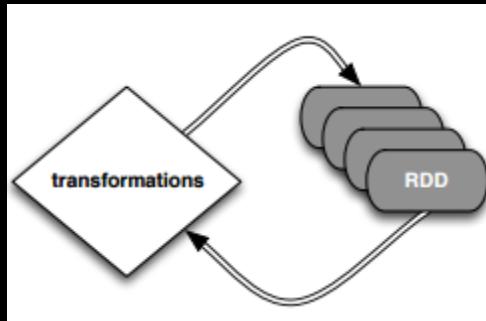


Operations – Step by Step



//base RDD

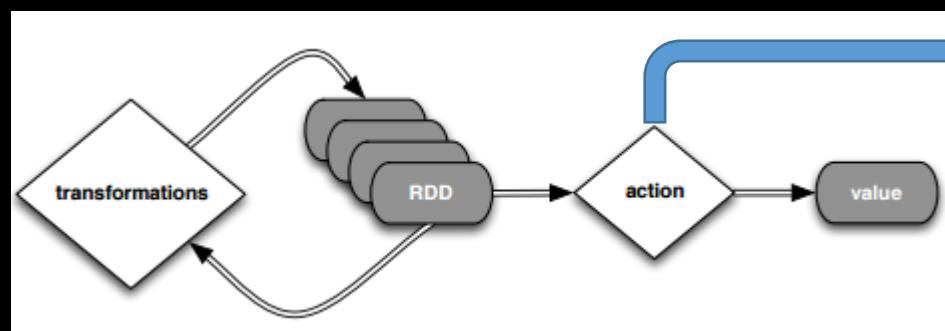
```
val lines = sc.textFile("hdfs://...")
```



//Transformed RDD

```
val errors = lines.filter(_.startsWith("Error"))
```

```
errors.persist()
```



errors.count()

count() causes Spark to:

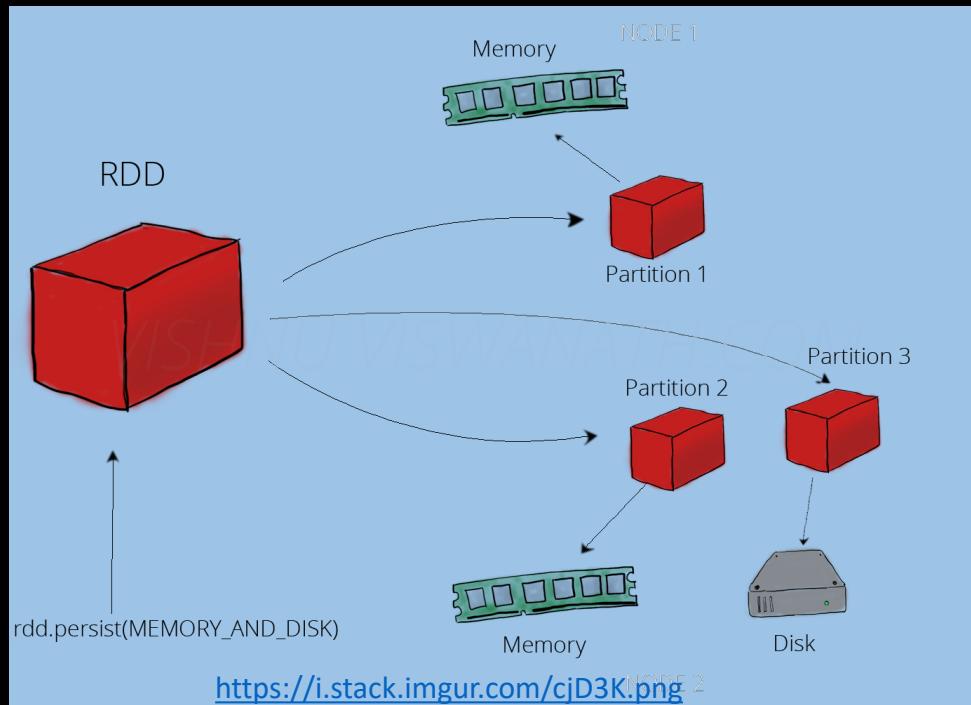
- 1) read data;
- 2) sum within partitions;
- 3) combine sums in driver

Put transform and action together:

```
errors.filter(_.contains("HDFS")).map(_ split ('\t')(3)).collect()
```

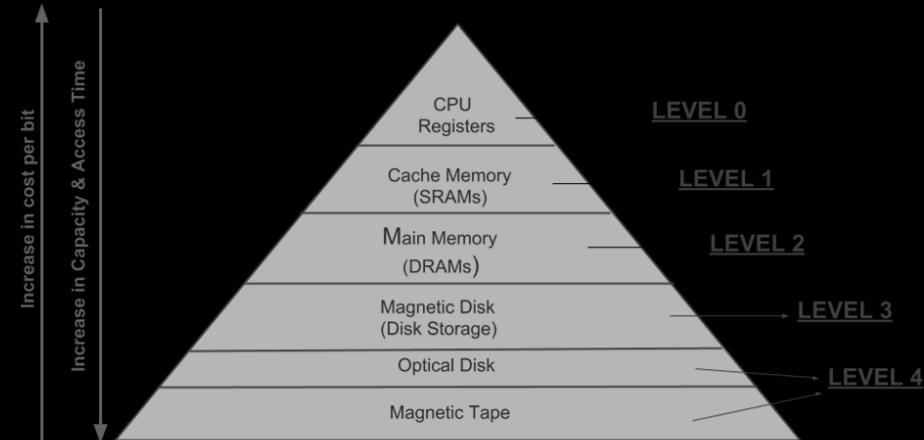
RDD Persistence

- Nodes store partitions for **reuse** in other actions on that dataset
- Storage levels for each persisted RDD
 - MEMORY_ONLY (**default**)
 - MEMORY_AND_DISK (**DataFrame default**)
 - Unfit partitions: to be recomputed when needed
- **cache() = persist(StorageLevel.MEMORY_ONLY)**



Why Persisting RDD?

```
val lines = sc.textFile("hdfs://...")  
  
val errors = lines.filter(_.startsWith("Error"))  
  
errors.persist()  
  
errors.count()
```



- `errors.count()` again → file reload and re-computation
- Persist → cache the data in memory → **reduce the data loading cost** for further actions on **the same data**
- `errors.persist()`: do nothing (a lazy operation, telling *"read this file and then cache the contents"*). An action will trigger computation and data caching.

Spark Key-Value RDDs

- Spark supports key-value pairs

groupByKey([numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.
reduceByKey(func, [numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <code>func</code> , which must be of type <code>(V,V) => V</code> . Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.
aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.
sortByKey([ascending], [numPartitions])	When called on a dataset of (K, V) pairs where K implements <code>Ordered</code> , returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean <code>ascending</code> argument.

Week 2 Contents / Objectives

- Resilient Distributed Datasets
- DataFrames and Datasets
- Machine Learning Pipelines
- Execution Parallelization

Why DataFrame?

- Challenges
 - ETL to/from various semi/unstructured data sources
 - Advanced analytics (e.g. machine learning) are hard to express in relational systems
- Solutions
 - A **DataFrame** API to perform relational operations on both external data sources and Spark's built-in RDDs
 - A highly extensible optimizer **Catalyst** to use Scala features to add composable rule, control code generation, and define extensions

DataFrame-based API for MLlib

- In v2.0, the DataFrame-based API became the primary API for MLlib
 - Voted by the community
 - `org.apache.spark.ml`, `pyspark.ml`
- The RDD-based API entered the maintenance mode
 - Still maintained with bug fixes, but no new features
 - `org.apache.spark.mllib`, `pyspark.mllib`

Announcement: DataFrame-based API is primary API

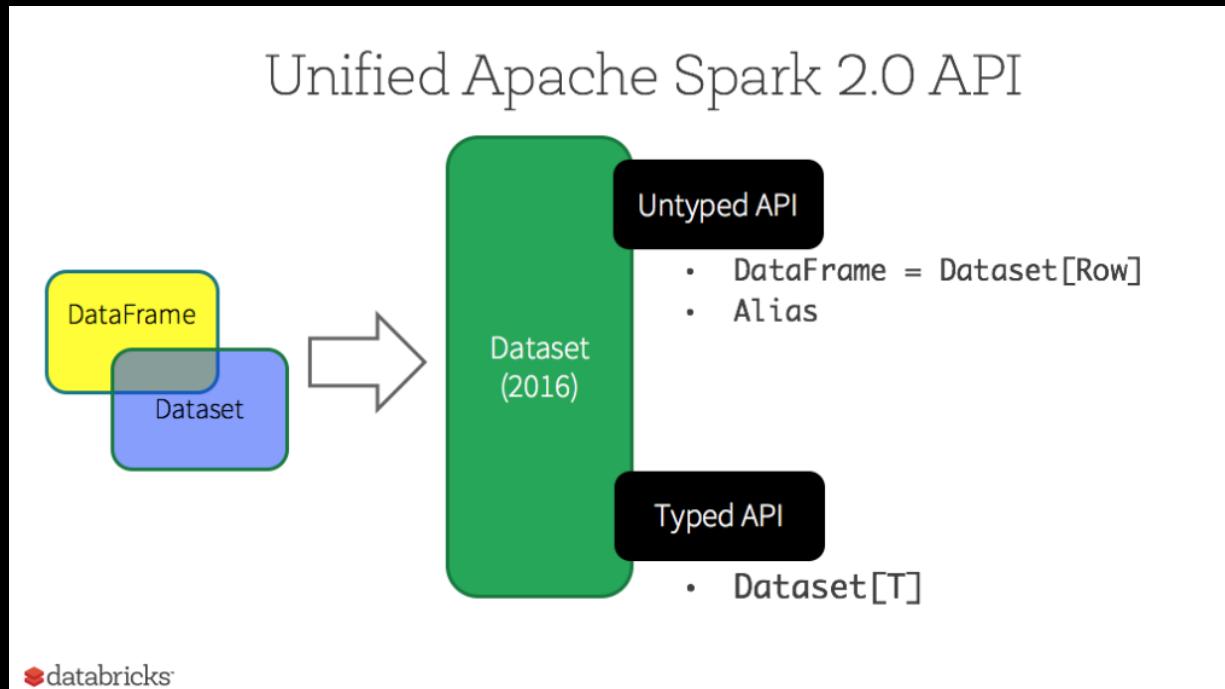
The MLlib RDD-based API is now in maintenance mode.

As of Spark 2.0, the [RDD-based APIs](#) in the `spark.mllib` package have entered maintenance mode. The primary Machine Learning API for Spark is now the [DataFrame-based API](#) in the `spark.ml` package.

What are the implications?

DataFrames and Datasets

- DataFrame: schema, generic untyped (like a table)
- Dataset: static typing, strongly-typed
- DataFrame = Dataset[Row] (Row: generic untyped)
 - Dataset organised into named columns



Typed and Un-typed APIs

Language	Main Abstraction
Scala	Dataset[T] & DataFrame (alias for Dataset[Row])
Java	Dataset[T]
Python*	DataFrame
R*	DataFrame

* Since Python and R have no compile-time type-safety,
we only have untyped APIs, namely DataFrames.

Benefits of Dataset APIs

- Static-typing and runtime type-safety
 - SQL least restrictive, no syntax error until runtime
 - DF/DS: syntax error detected at compile time
- High-level abstraction and custom view into structured and semi-structured data, e.g. CSV
- Ease-of-use of APIs with structure
 - Rich semantics and domain specific operations
- Performance and optimization
 - SQL Catalyst

DataFrame

- A distributed collection of rows with the same schema
- Can be constructed from external data sources or RDDs into essentially an RDD of Row objects
- Supports relational operators (e.g. **where**, **groupBy**) as well as Spark operations

dept	age	name
Bio	48	H Smith
CS	54	A Turing
Bio	43	B Jones
Chem	61	M Kennedy

RDD API

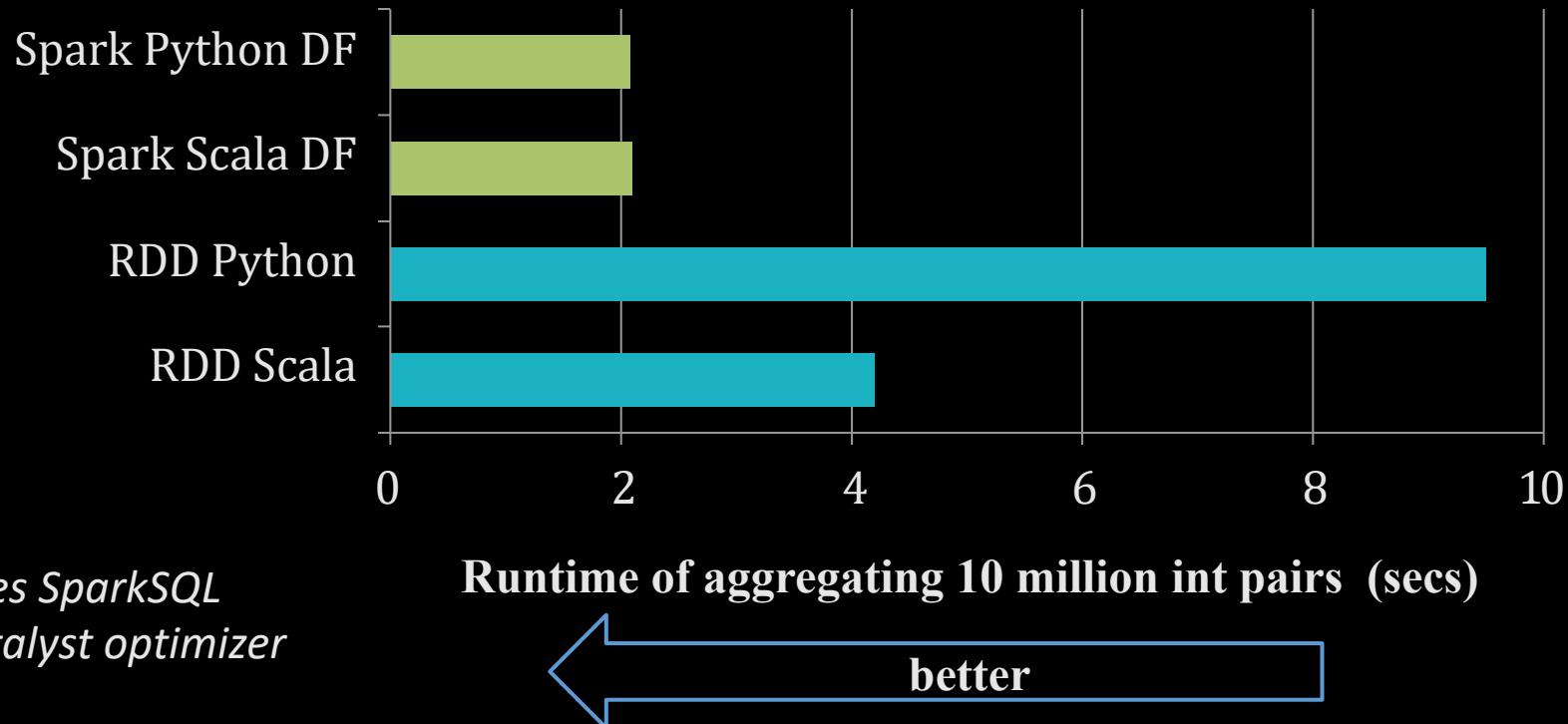
```
pdata.map(lambda x: (x.dept, [x.age, 1])) \  
    .reduceByKey(lambda x, y: [x[0] + y[0], x[1] + y[1]]) \  
    .map(lambda x: [x[0], x[1][0] / x[1][1]]) \  
    .collect()
```

DataFrame API

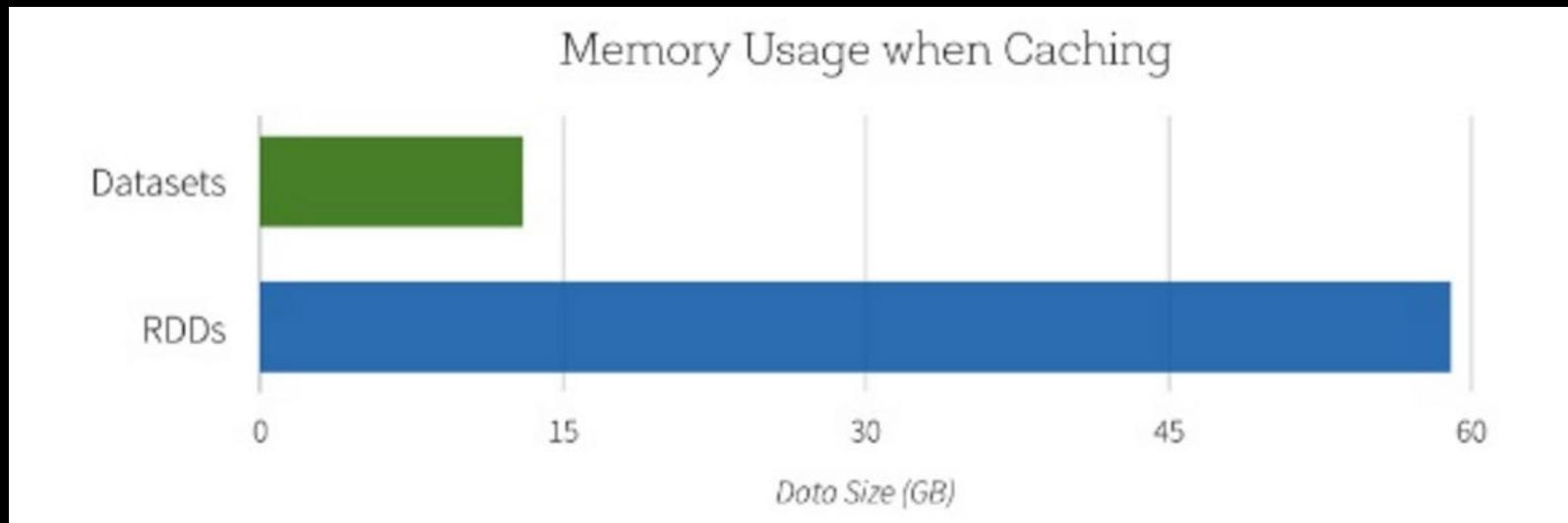
```
data.groupBy("dept").avg("age")
```

Data grouped into
named columns

Spark DataFrames are Fast



Space Efficiency



Week 2 Contents / Objectives

- Resilient Distributed Datasets
- DataFrames and Datasets
- Machine Learning Pipelines
- Execution Parallelization

Machine Learning Library (MLlib)

- **ML algorithms**: common ML algorithms for regression, classification, clustering, and collaborative filtering
- **Featurization**: feature extraction, transformation, dimensionality reduction, and selection
- **Pipelines**: tools for constructing, evaluating, and tuning ML pipelines
- **Persistence**: save/load algorithms, models, & pipelines
- **Utilities**: linear algebra, statistics, data handling, ...

Main Concepts in Pipelines

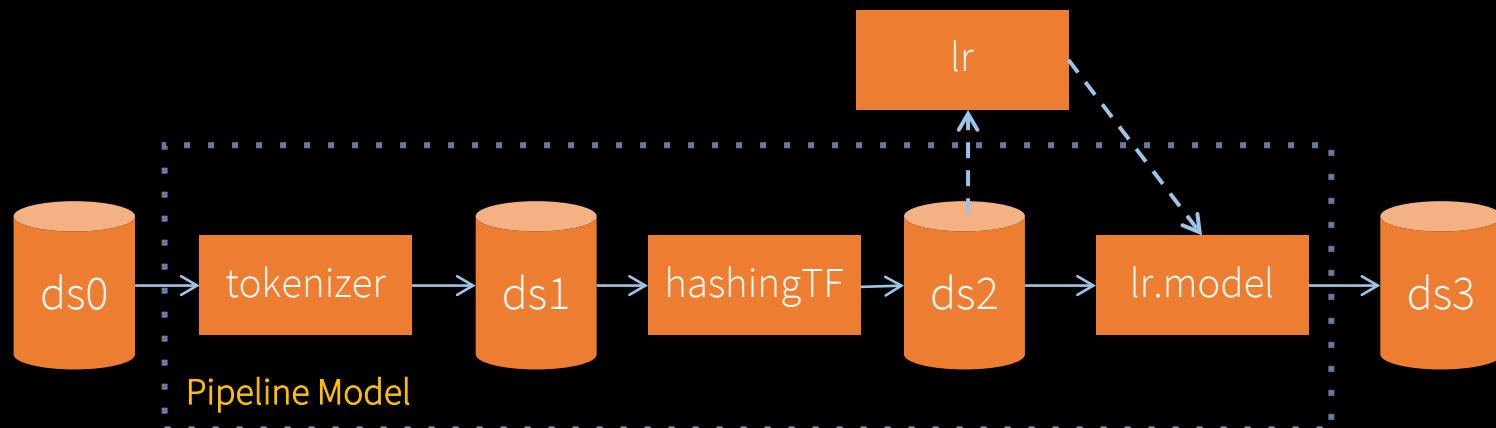
- **DataFrame**: an ML dataset holding various data types, e.g. columns for text, feature vectors, true labels, & predictions
- **Transformer**: algorithm transforming one DataFrame into another, e.g. features → **ML model** → predictions
- **Estimator**: algorithm fitting on a DataFrame to produce a Transformer, e.g. training data → **ML algorithm** → ML model
- **Pipeline**: chains multiple Transformers and Estimators together to specify an ML workflow
- **Parameter**: all Transformers and Estimators now share a common API for specifying parameters

ML Pipelines

- High-level APIs to create and tune ML pipelines

```
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol="words", outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.01)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])
```

```
df = spark.read.load("/path/to/data")
model = pipeline.fit(df)
```



Example: Text Classification

Goal: Given a text document, predict its topic.

Features

Subject: Re: Lexan Polish?
Suggest McQuires #1 plastic
polish. It will help somewhat
but nothing will remove deep
scratches without making it
worse than it already is.
McQuires will do
something...

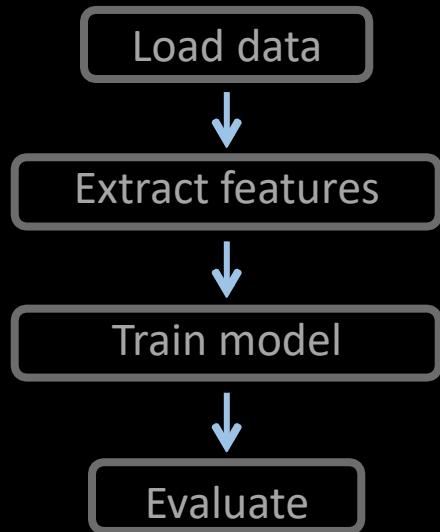


Label

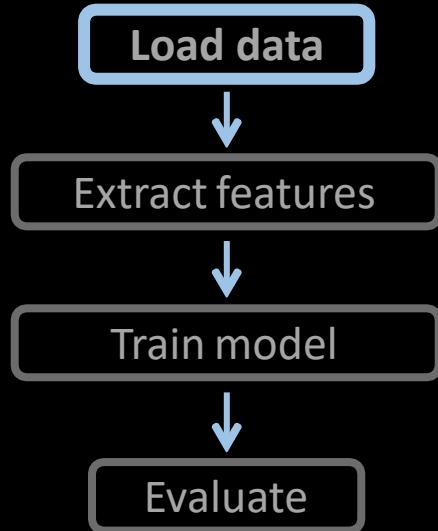
1: about science
0: not about science

Dataset: “20 Newsgroups”
From UCI KDD Archive

ML Workflow



Load Data



Current data schema

label: Int

text: String

Data sources for DataFrames

built-in



external



Extract Features

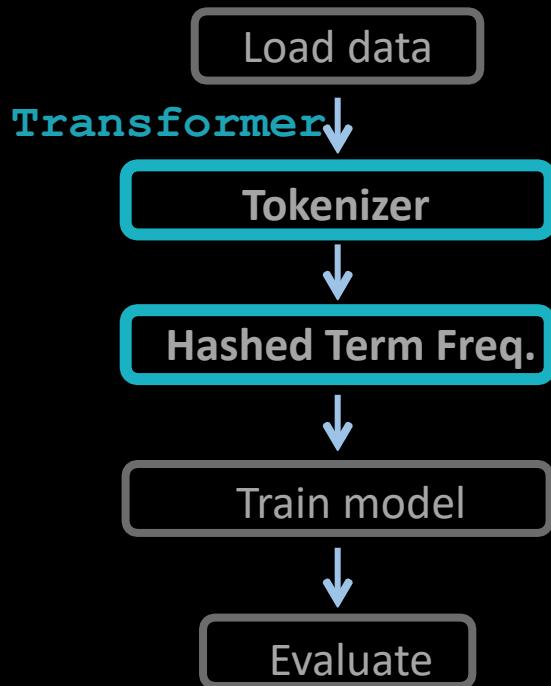


Current data schema

label: Int

text: String

Extract Features



Current data schema

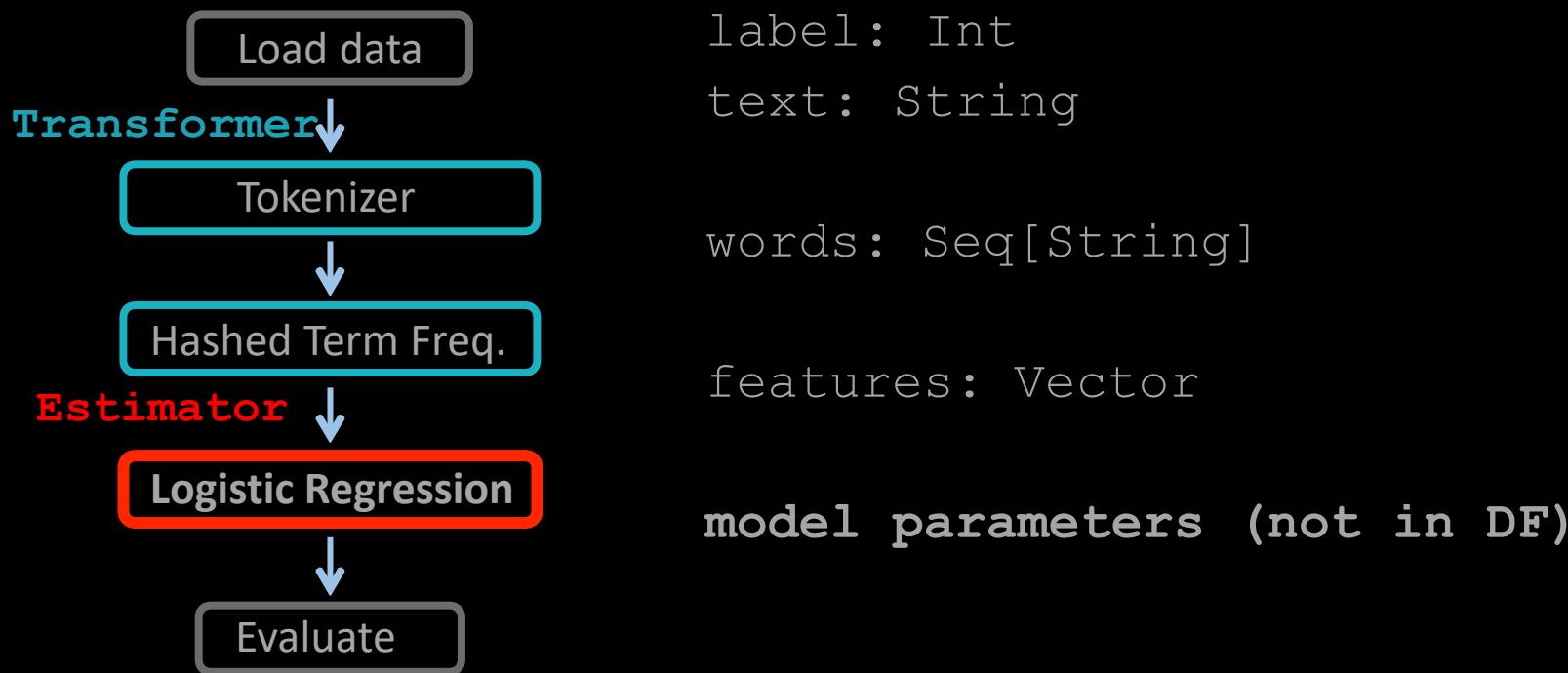
label: Int

text: String

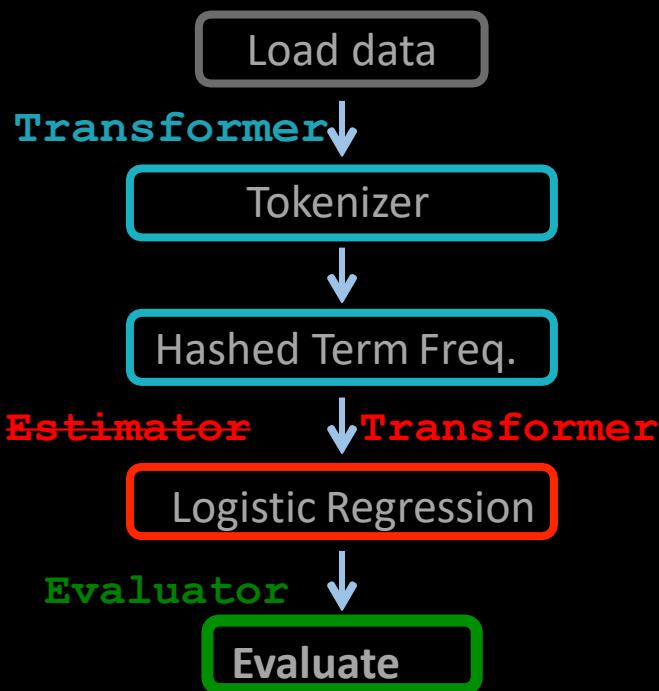
words: Seq[String]

features: Vector

Train the Model



Evaluate the Model



Current data schema

label: Int

text: String

words: Seq[String]

features: Vector

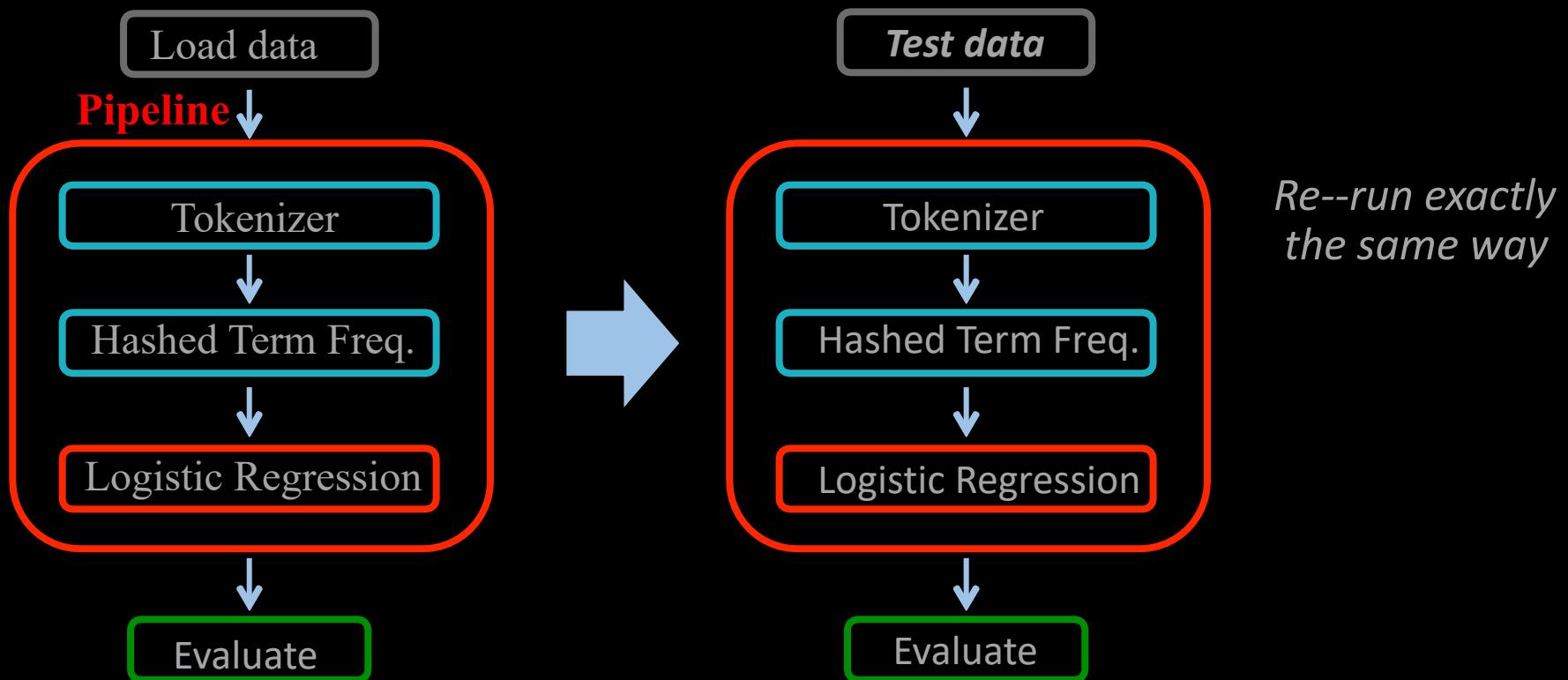
prediction: Int

By default, always append new columns

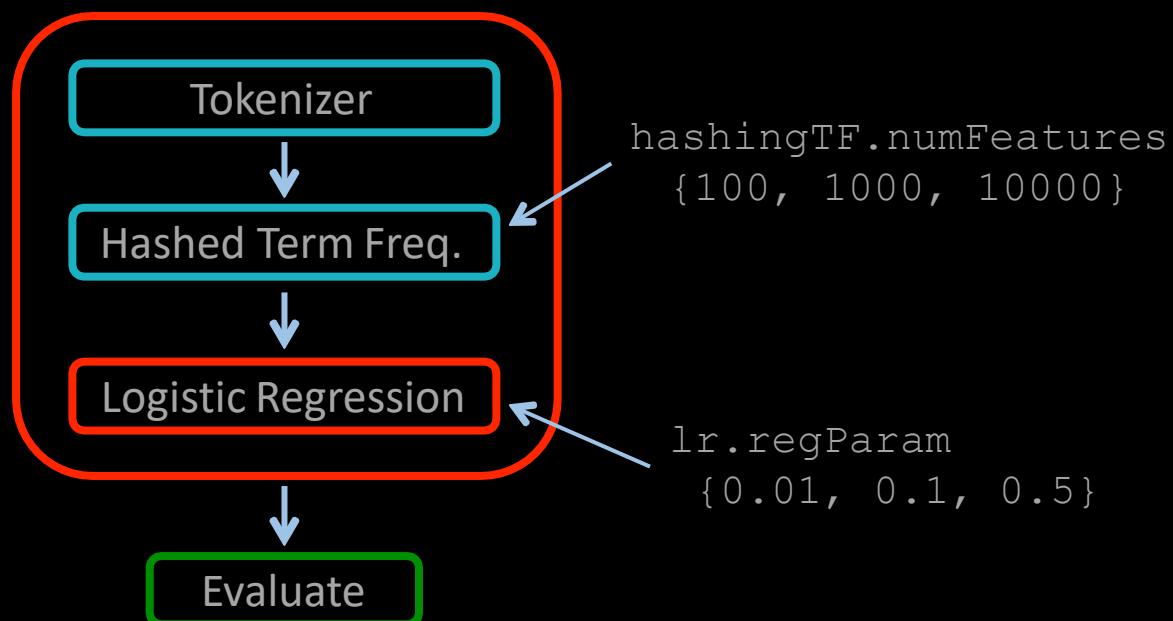
→ Can go back & inspect intermediate results

→ Made efficient by DataFrame optimizations

ML Pipelines



Parameter Tuning



CrossValidator

Given:

- Estimator
- Parameter grid
- Evaluator

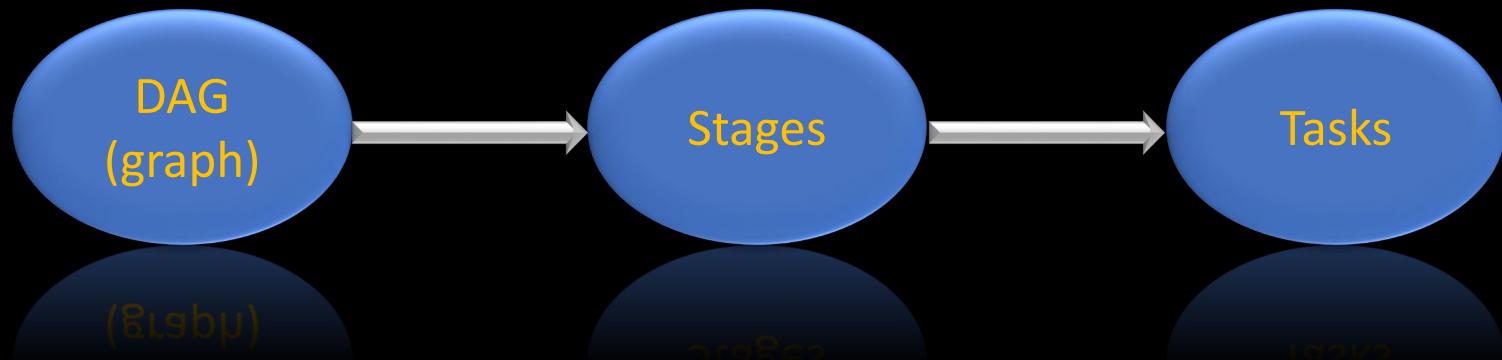
Find best parameters

Week 2 Contents / Objectives

- Resilient Distributed Datasets
- DataFrames and Datasets
- Machine Learning Pipelines
- Execution Parallelization

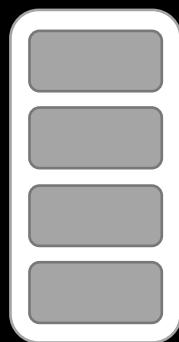
How Spark Works

- User applications create RDDs/DFs, transform them, and run actions
- This results in a **DAG** (Directed Acyclic Graph) of operators
- DAG is compiled into **stages**
- Each stage is executed as a series of **tasks**

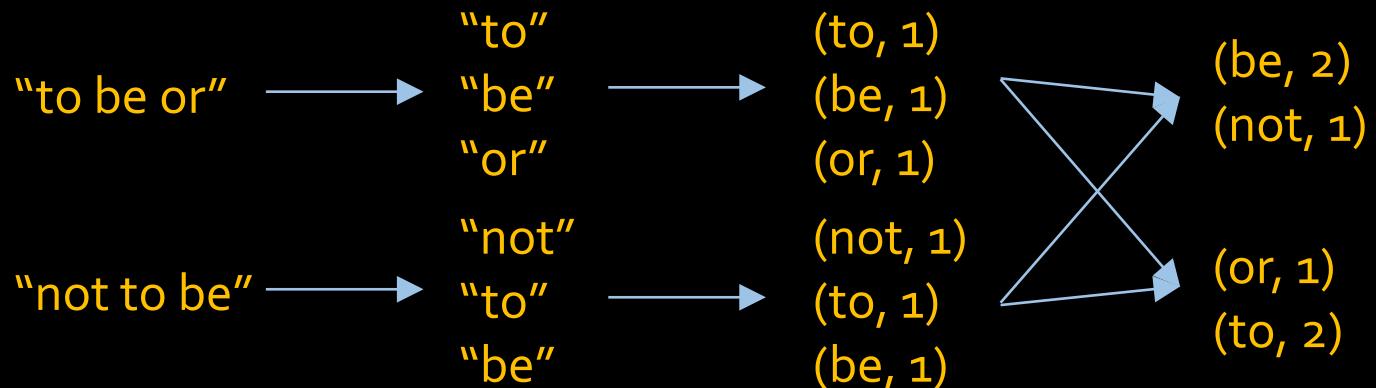


Word Count in Spark

```
val file = sc.textFile("hdfs://...", 4) RDD[String]
```

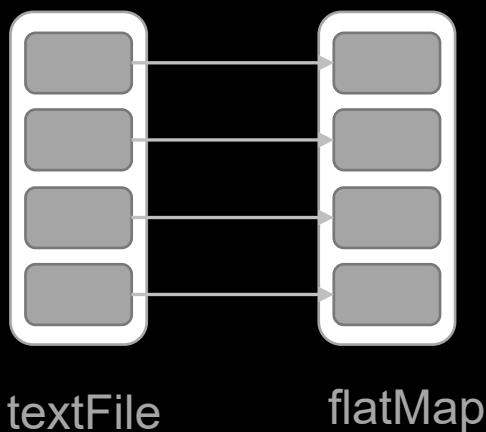


textFile



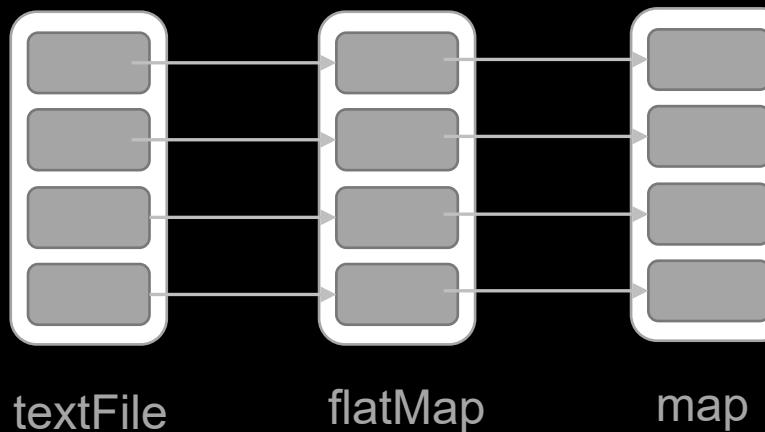
Word Count in Spark

```
val file = sc.textFile("hdfs://...", 4)          RDD[String]  
val words = file.flatMap(line =>  
    line.split("\t"))                           RDD[List[String]]
```



Word Count in Spark

```
val file = sc.textFile("hdfs://...", 4)          RDD[String]  
val words = file.flatMap(line =>  
    line.split("\t"))  
val pairs = words.map(t => (t, 1))            RDD[(String, Int)]
```

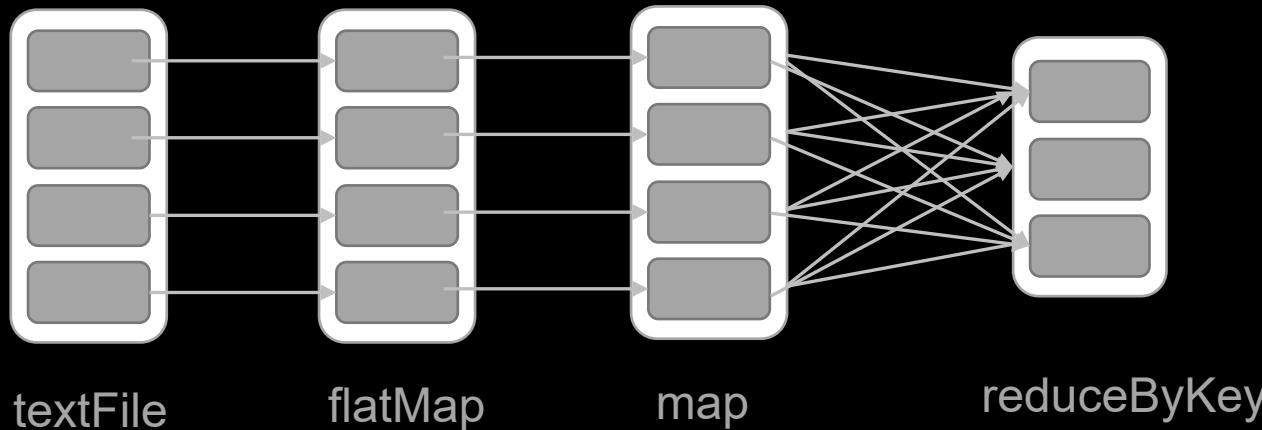


Word Count in Spark

```
val file = sc.textFile("hdfs://...", 4)          RDD[String]  
val words = file.flatMap(line =>  
    line.split("\t"))                            RDD[List[String]]  
  
val pairs = words.map(t => (t, 1))            RDD[(String, Int)]  
val count = pairs.reduceByKey(_+_)
```

RDD[(String, Int)]

RDD[(String, Int)]

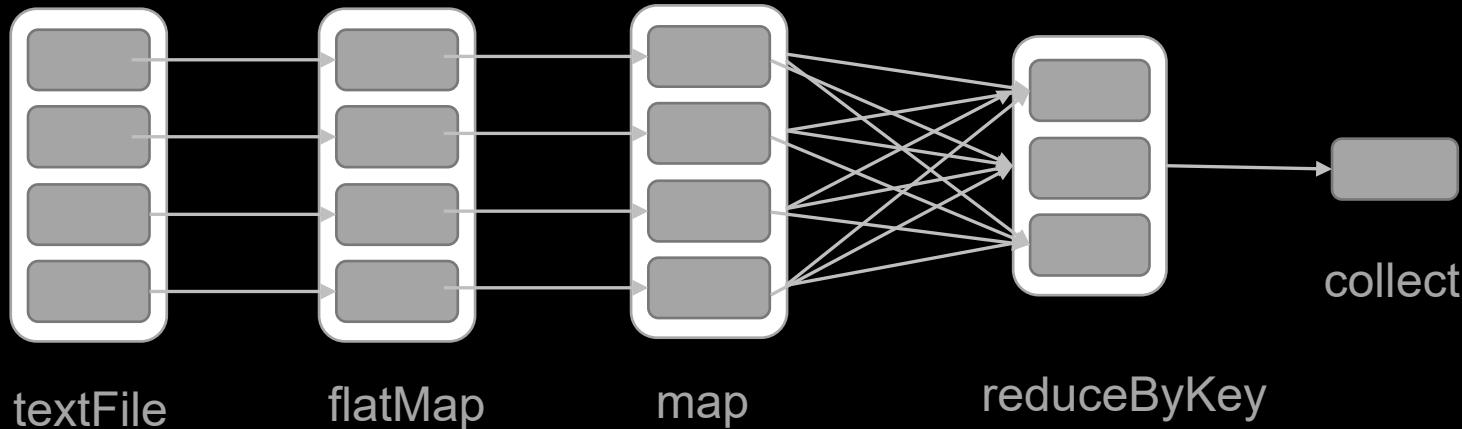


Word Count in Spark

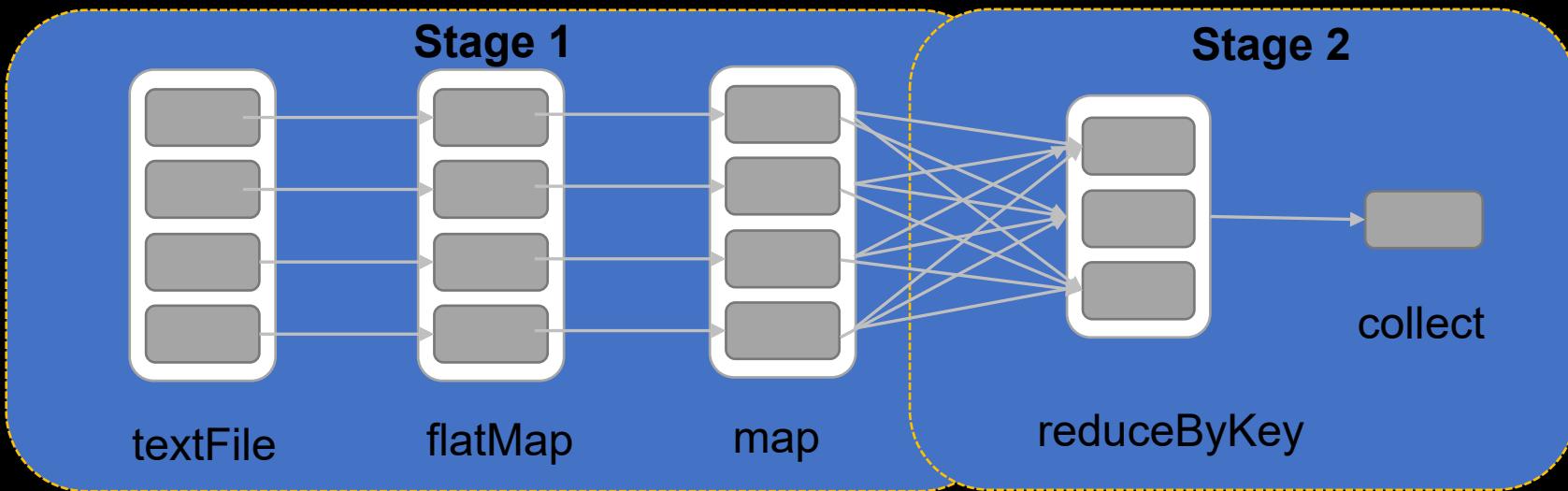
```
val file = sc.textFile("hdfs://...", 4)          RDD[String]  
val words = file.flatMap(line =>  
    line.split("\t"))                            RDD[List[String]]  
  
val pairs = words.map(t => (t, 1))            RDD[(String, Int)]  
val count = pairs.reduceByKey(_+_)
```

count.collect()

```
RDD[(String, Int)]  
RDD[(String, Int)]  
Array[(String, Int)]
```

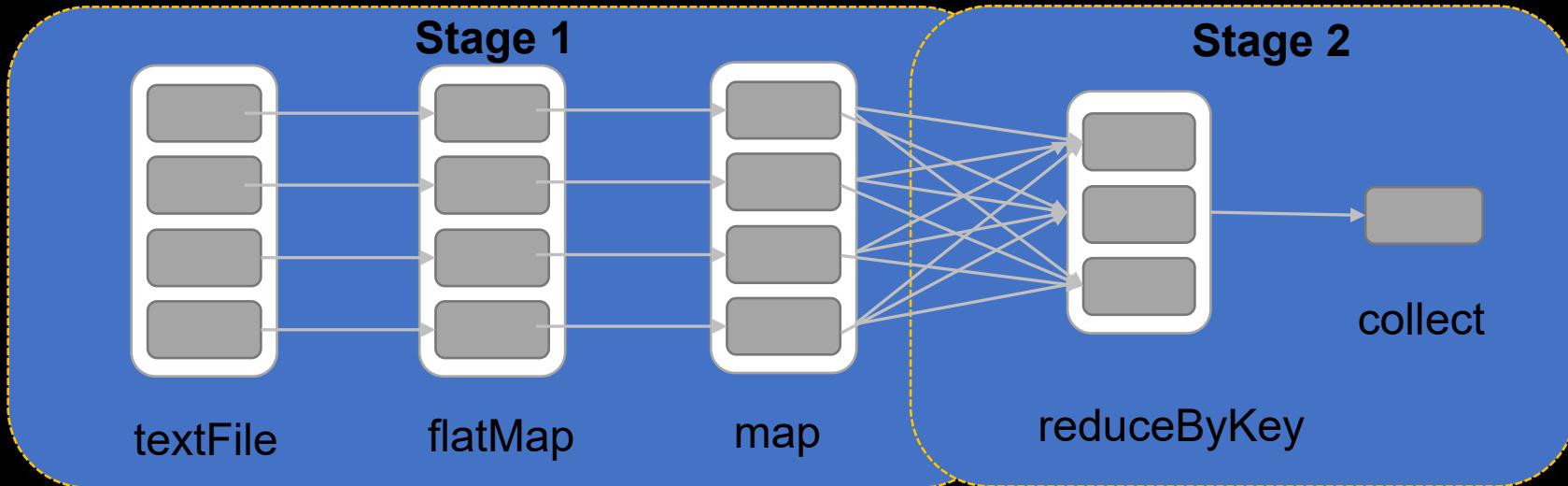


Execution Plan

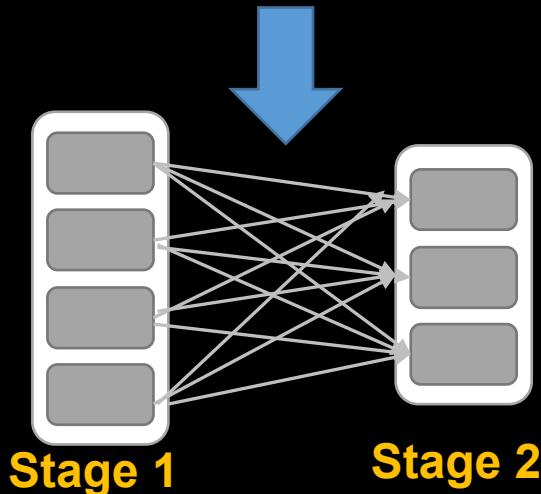


- The scheduler examines the RDD's lineage graph to build a DAG of stages
- Stages are sequences of RDDs, that don't have a **shuffle** in between

Execution Plan



1. Read HDFS split
2. Apply both the maps
3. Start partial reduce
4. Write shuffle data



1. Read shuffle data
2. Final reduce
3. Send result to driver program

Execution of Tasks



- Create a **task** for each partition in the new RDD
- Compute the task's **closure** (those variables and methods that must be visible to the worker)
- Serialize the task's closure
- Schedule and ship tasks (closures) to workers

Setting the Level of Parallelism

- Many transformations take an optional parameter **numPartitions** for number of tasks

<code>distinct([numPartitions])</code>	Return a new dataset that contains the distinct elements of the source dataset.
<code>groupByKey([numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, Iterable<V>) pairs. Note: If you are grouping in order to perform an aggregation (such as a sum or average) over each key, using <code>reduceByKey</code> or <code>aggregateByKey</code> will yield much better performance. Note: By default, the level of parallelism in the output depends on the number of partitions of the parent RDD. You can pass an optional <code>numPartitions</code> argument to set a different number of tasks.
<code>reduceByKey(func, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, V) pairs where the values for each key are aggregated using the given reduce function <code>func</code> , which must be of type (V,V) => V. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.
<code>aggregateByKey(zeroValue)(seqOp, combOp, [numPartitions])</code>	When called on a dataset of (K, V) pairs, returns a dataset of (K, U) pairs where the values for each key are aggregated using the given combine functions and a neutral "zero" value. Allows an aggregated value type that is different than the input value type, while avoiding unnecessary allocations. Like in <code>groupByKey</code> , the number of reduce tasks is configurable through an optional second argument.
<code>sortByKey([ascending], [numPartitions])</code>	When called on a dataset of (K, V) pairs where K implements Ordered, returns a dataset of (K, V) pairs sorted by keys in ascending or descending order, as specified in the boolean

Shared Variables (for Cluster)

- Variables are distributed to workers via closures
- When a function is executed on a cluster node, it works on **separate** copies of those variables that are not shared across workers
- **Iterative** or single jobs with large global variables
 - **Problem:** inefficient to send large data with each iteration
 - Solution: Broadcast variables (keep rather than ship)
- Counting events that occur **during** job execution
 - **Problem:** Closures are one way driver → worker
 - Solution: Accumulators (only “added” to, e.g. sums/counters)

Recommended Reading

- [A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets](#)
- [Sections 2.4.2 and 2.4.3 of the MMDS book \(3rd edition\)](#)
- Hyperlinks in slides
- Suggested reading in Lab 2

Scalable decision trees and ensembles

Mauricio A. Álvarez

Scalable Machine Learning,
University of Sheffield



The
University
Of
Sheffield.

Decision trees and ensembles in MLAI

- Last semester, as part of the MLAI module, we studied decision trees and ensemble methods.
- There was also a Jupyter Notebook and an Assignment looking at how to use `scikit-learn` to apply these ML methods to actual data.
- In this lecture, we will not explain decision trees and ensemble methods to the same level of detail but will review some relevant aspects necessary to use both models in PySpark.
- The APIs that we use in `spark.ml` are very similar to the ones used in `scikit-learn`, with subtle differences.

Why do we like decision trees?

- easy to interpret.
- handle categorical features.
- extend to the multiclass classification setting.
- do not require feature scaling.
- able to capture non-linearities and feature interactions.

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

Nodes in a decision tree

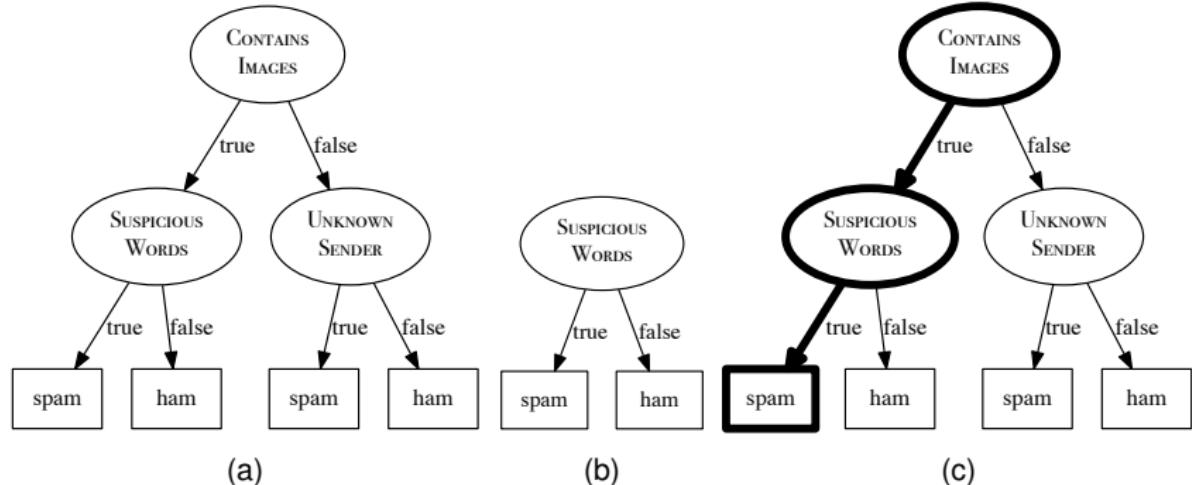
- A decision tree consists of:
 1. a **root node** (or starting node),
 2. **interior nodes**
 3. and **leaf nodes** (or terminating nodes).
- Each of the non-leaf nodes (root and interior) in the tree specifies a test to be carried out on one of the query's descriptive features.
- Each of the leaf nodes specifies a predicted classification or predicted regression value for the query.

Binary classification: spam prediction

An email spam prediction dataset.

ID	SUSPICIOUS WORDS	UNKNOWN SENDER	CONTAINS IMAGES	CLASS
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

Two decision trees and a query instance

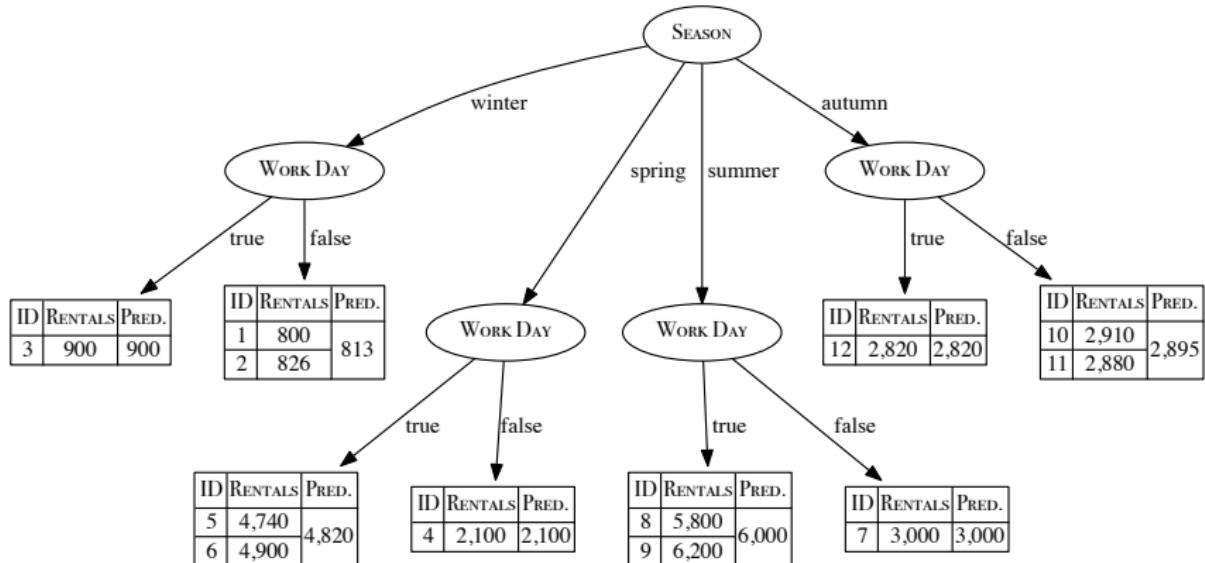


(a) and (b) show two decision trees that are consistent with the instances in the spam dataset. (c) shows the path taken through the tree shown in (a) to make a prediction for the query instance: SUSPICIOUS WORDS = 'true', UNKNOWN SENDER = 'true', CONTAINS IMAGES = 'true'.

Regression: bike rentals per day

ID	SEASON	WORK DAY	RENTALS
1	winter	false	800
2	winter	false	826
3	winter	true	900
4	spring	false	2 100
5	spring	true	4 740
6	spring	true	4 900
7	summer	false	3 000
8	summer	true	5 800
9	summer	true	6 200
10	autumn	false	2 910
11	autumn	false	2 880
12	autumn	true	2 820

Regression tree



The final decision tree induced from the dataset. This tree lists the instances that ended up at each leaf node and the prediction (PRED.) made by each leaf node.

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

Binary decision trees

- ❑ Apache Spark only builds binary decision trees: each node can only have two branches.
- ❑ Binary partitions at each node are done recursively.
- ❑ Each partition is chosen greedily by selecting the *best split* from a set of possible splits.

Best split

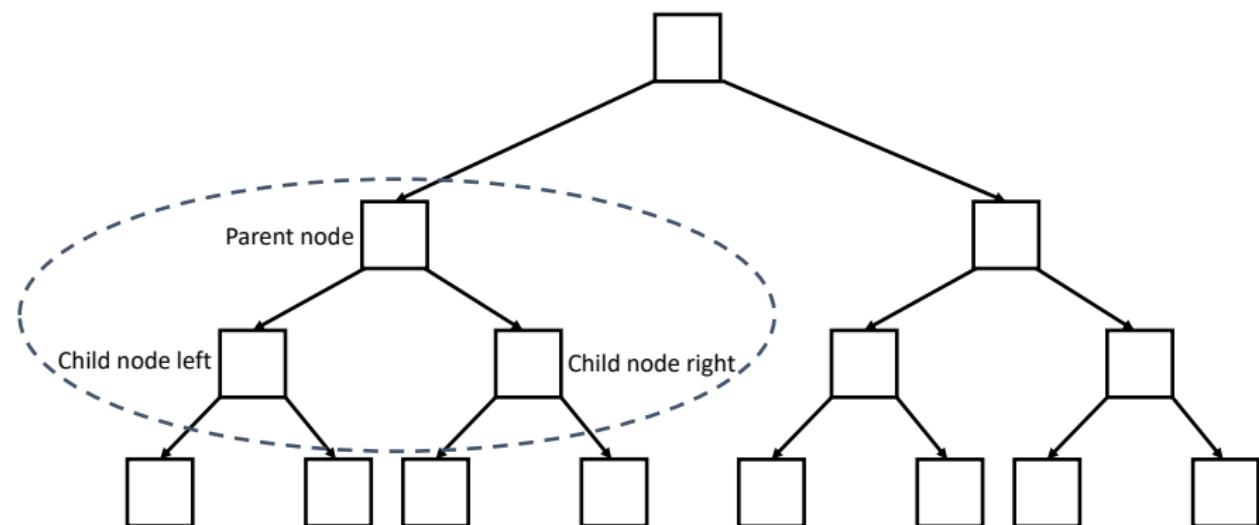
- The best split maximizes the *information gain* at a tree node.
- The split chosen at each tree node is chosen from the set

$$\arg \max_s IG(D, s),$$

where $IG(D, s)$ is the information gain when split s is applied to dataset D .

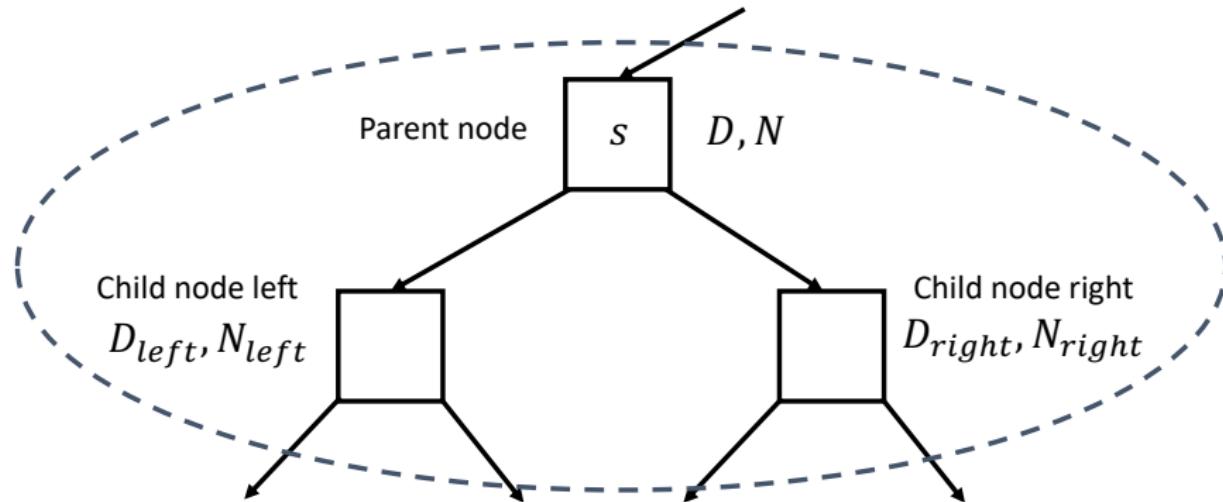
Information gain

The *information gain* is the difference between the parent node impurity and the weighted sum of the two child node impurities.



Information gain

Assuming that a split s partitions the dataset D of size N into two datasets D_{left} and D_{right} of sizes N_{left} and N_{right} ,



the information gain is computed as

$$IG(D, s) = \text{Impurity}(D) - \left(\frac{N_{\text{left}}}{N} \text{Impurity}(D_{\text{left}}) + \frac{N_{\text{right}}}{N} \text{Impurity}(D_{\text{right}}) \right).$$

Node impurity

- Remember that the node impurity is a measure of how homogeneous are the labels at a node.
- The table below shows the impurity measures currently implemented in Spark

Impurity	Task	Formula	Description
Gini impurity	Classification	$\sum_{i=1}^C f_i(1 - f_i)$	f_i is the frequency of label i at a node and C is the number of unique labels.
Entropy	Classification	$\sum_{i=1}^C -f_i \log(f_i)$	f_i is the frequency of label i at a node and C is the number of unique labels.
Variance	Regression	$\frac{1}{N} \sum_{i=1}^N (y_i - \mu)^2$	y_i is label for an instance, N is the number of instances and μ is the mean given by $\frac{1}{N} \sum_{i=1}^N y_i$.

Frequency refers to probability. Link [here](#)

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

How are the split candidates chosen?

- ❑ We saw before that Spark only implements binary decision trees.
- ❑ This means the split is always of the kind:

```
if feature  $i \leq$  threshold then
    move to the left branch (or the right branch)
else
    move to the right branch (or the left branch)
end if
```
- ❑ So, how is threshold computed? It depends on whether the feature is continuous or categorical.

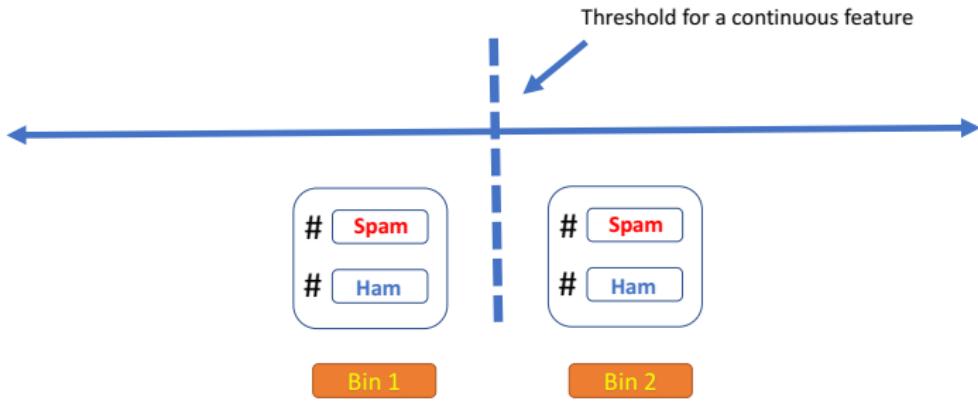
Handling continuous features

- When working with small datasets, we can either:
 - directly use the values of the feature as the thresholds.
 - sort the values of the feature and compute the corresponding thresholds from the sorted values.
- This is what we saw in MLAI.
- However, these alternatives are not possible to use for large distributed datasets
 - it is not feasible to use the values of the feature as the thresholds.
 - although sorting is an option, it is expensive.

Handling continuous features

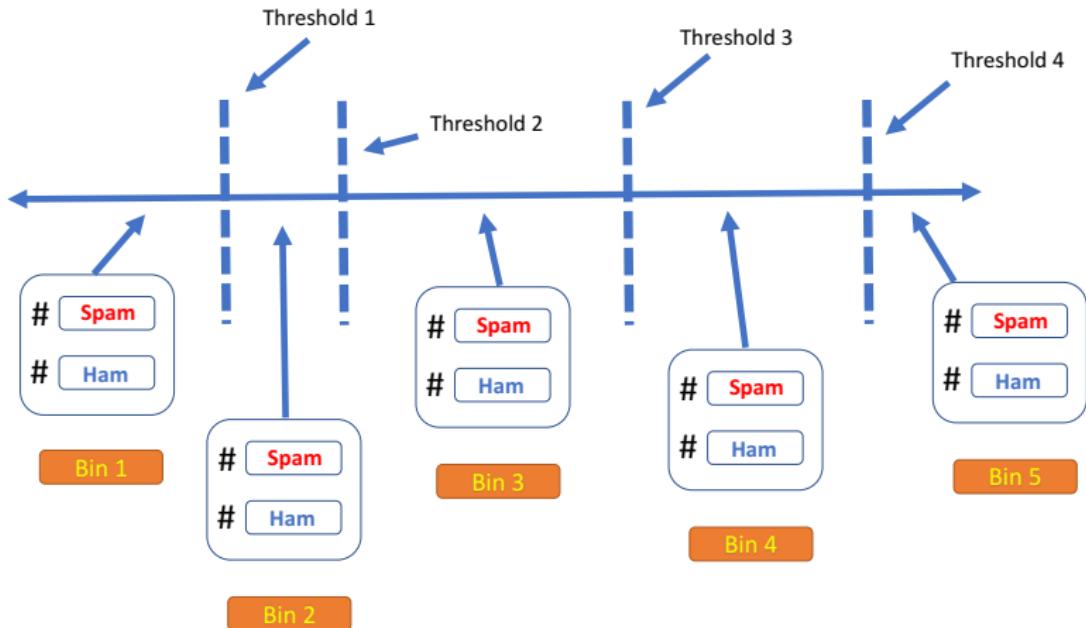
- ❑ Spark's implementation computes an approximate set of split candidates by performing a quantile calculation over a sampled fraction of the data.
- ❑ The ordered splits create “bins” and the maximum number of such bins can be specified using the `maxBins` parameter.

Bins for continuous features



Bins in continuous features

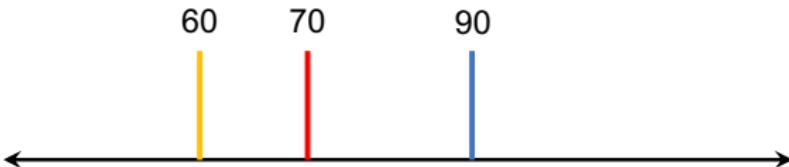
Bins for continuous features



Bins in continuous features

Why bins? Naive approach

ID	Frequency of word “free”	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



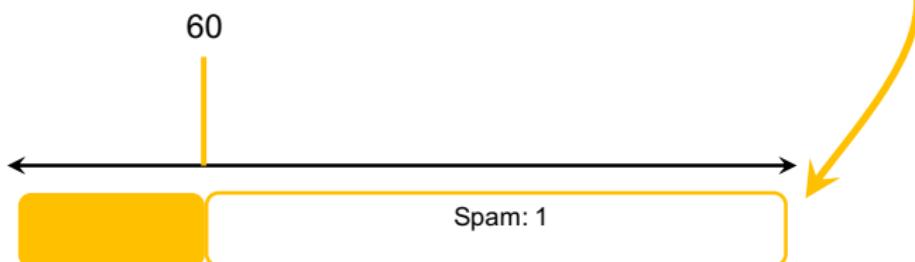
Why bins? Naive approach

ID	Frequency of word “free”	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



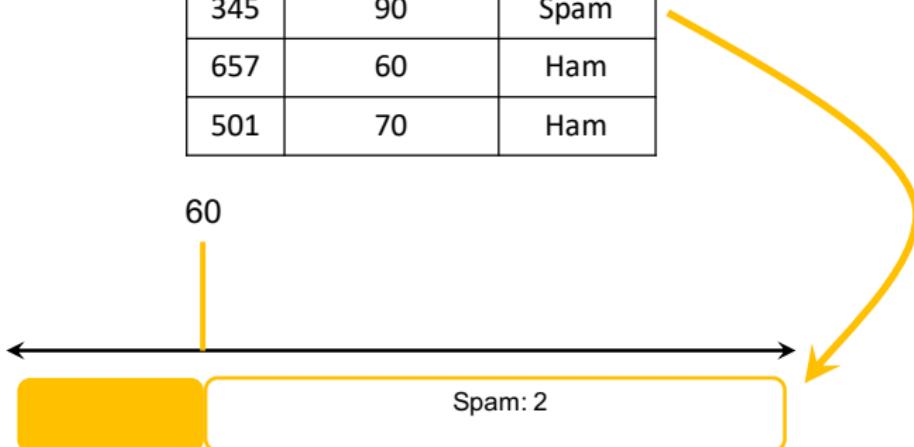
Why bins? Naive approach

ID	Frequency of word “free”	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



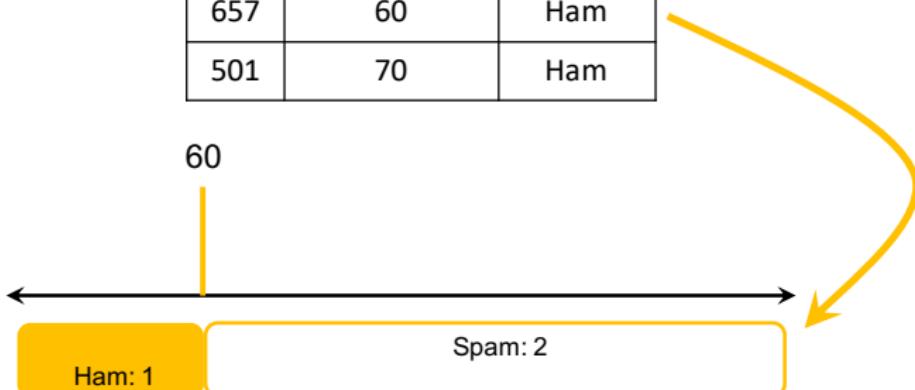
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



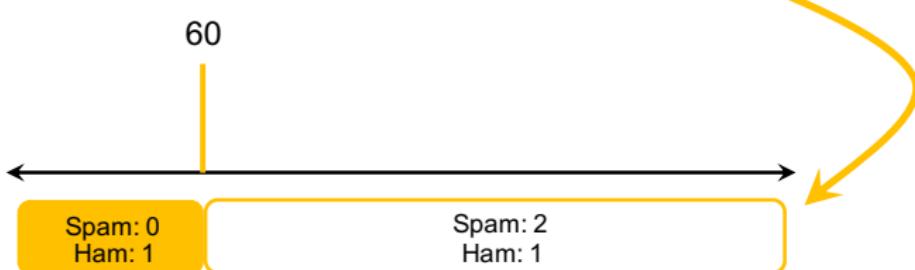
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



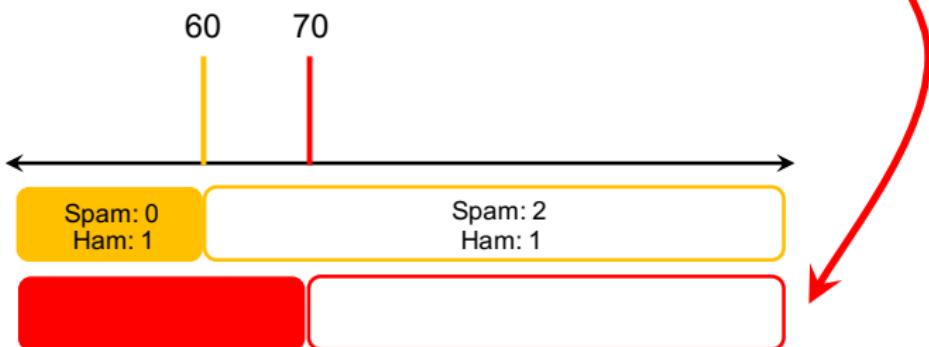
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



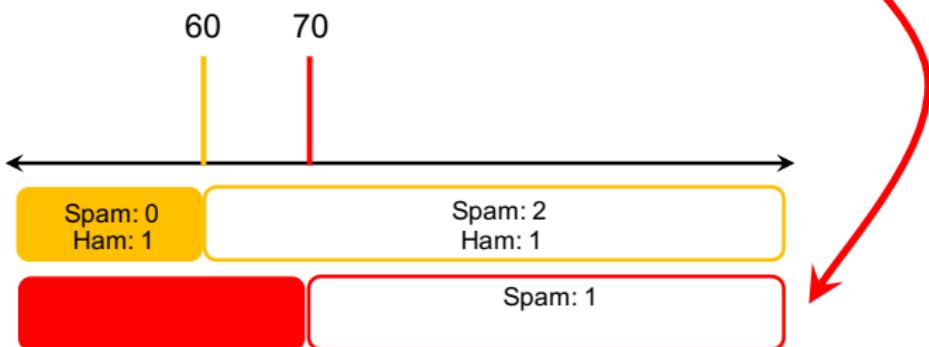
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



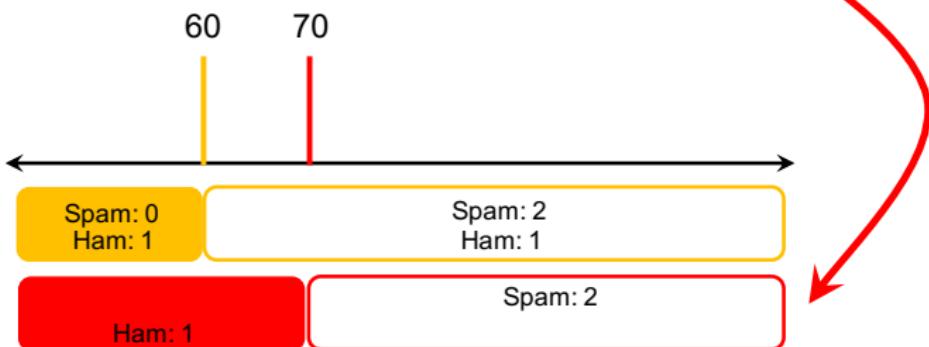
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



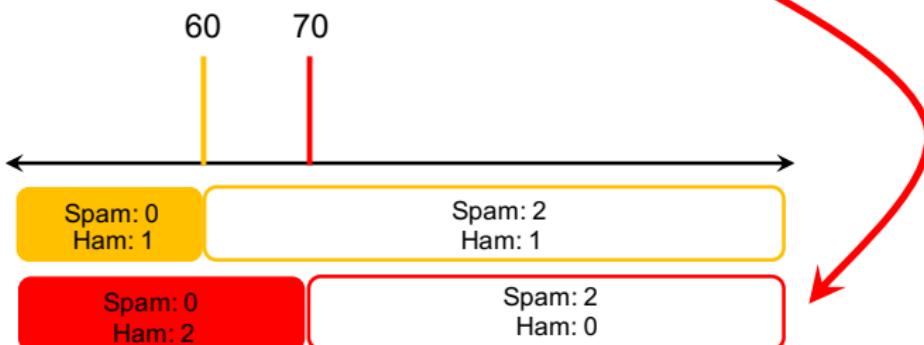
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



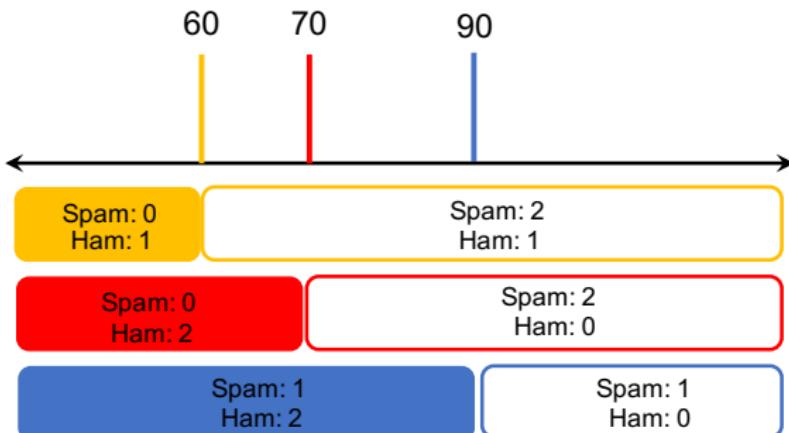
Why bins? Naive approach

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



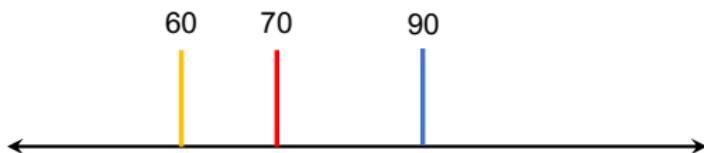
Why bins? Naive approach

ID	Frequency of word “free”	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



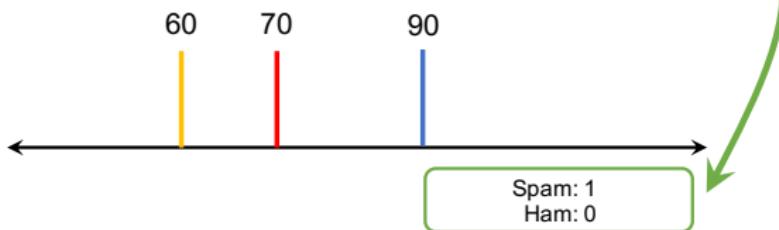
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



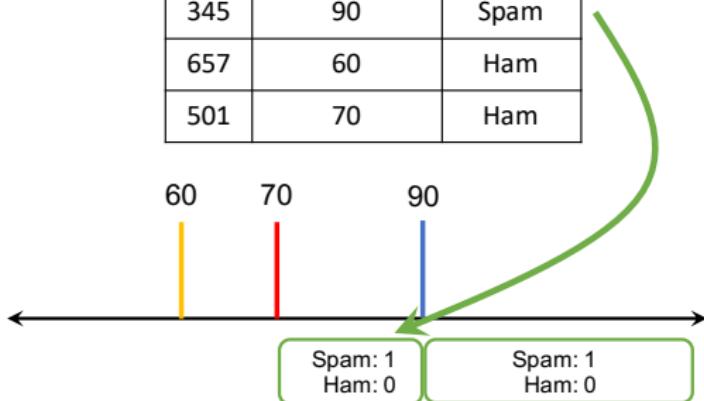
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



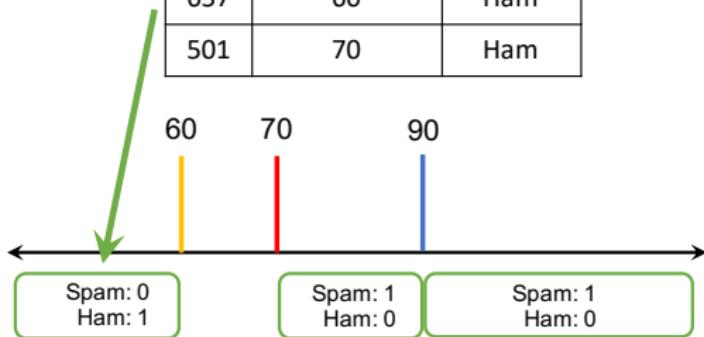
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



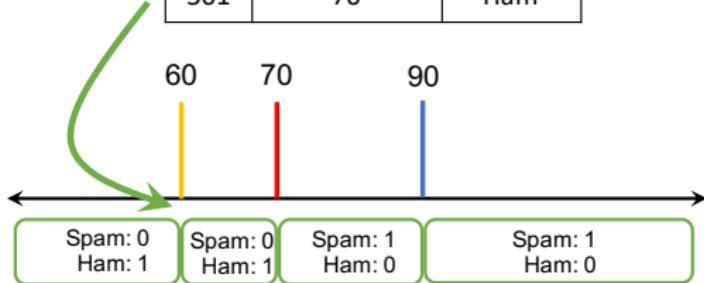
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



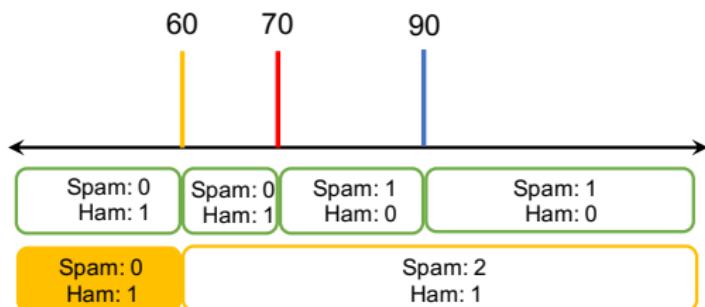
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



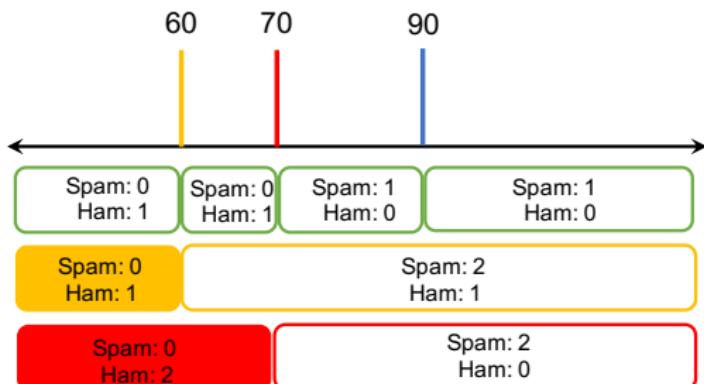
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



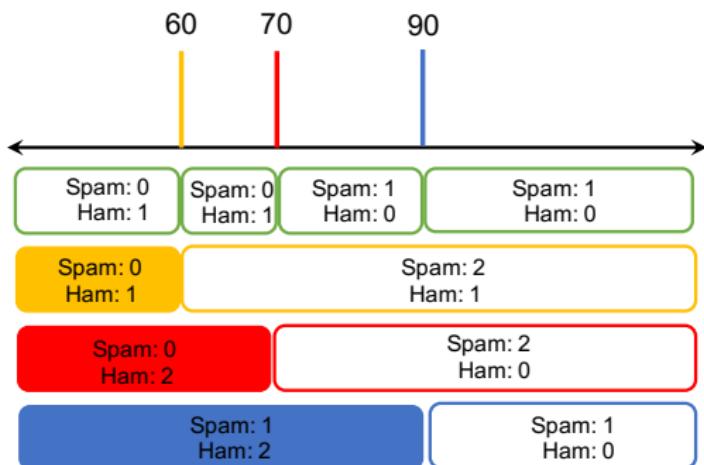
Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



Why bins? Binary search

ID	Frequency of word "free"	Class
908	95	Spam
345	90	Spam
657	60	Ham
501	70	Ham



What is the difference?

- Assume there are m thresholds.
- In the naive approach, for each observation, we need to perform m operations.
- When using binning, it is possible to use binary search for each observation to fill out the bins.
- Binary search takes $\log(m)$ operations.

Categorical features

- ❑ Say we have a categorical feature that can take M unordered values.
- ❑ We need to make a binary partition based on this categorical feature.
- ❑ It can be shown there are $2^{M-1} - 1$ possible partitions of the M values into two groups.

Categorical features

- As an example, consider the feature `weather` that takes values [spring, summer, autumn, winter]
- We then have $2^{M-1} - 1 = 2^{4-1} - 1 = 7$ possible partitions of two groups

Option	Group 1	Group 2
1	spring	summer, autumn, winter
2	summer	spring, autumn, winter
3	autumn	summer, spring, winter
4	winter	summer, autumn, spring
5	spring, summer	autumn, winter
6	spring, autumn	summer, winter
7	spring, winter	summer, autumn

- For M large, computation becomes prohibitive.
- There are heuristics to optimally reduce the number of partitions depending on the type of prediction problem.

Categorical features: binary classification

- For binary classification, the number of partitions can be reduced to $M - 1$ by ordering the categorical feature values by the proportion falling in outcome class 1.
- For example, for the feature `weather` say that the proportions for outcome class 1 according to the categorical values are

Categorical value	Proportion
spring	0.2
summer	0.3
autumn	0.05
winter	0.45

- Ordering the categorical values, we get [autumn, spring, summer, winter] and the $M - 1 = 3$ partitions are

Option	Group 1	Group 2
1	autumn	spring, summer, winter
2	autumn, spring	summer, winter
3	autumn, spring, summer	winter

- Essentially, we are transforming an unordered feature into an ordered feature.

Categorical features: regression

- One can show this gives the optimal split, in terms of cross-entropy or Gini index, among all possible $2^{M-1} - 1$ splits.
- This result also holds for a regression problem when using the square error loss.
- In this case, the categories are ordered by increasing mean of the outcome.
- For more details, see [The Elements of Statistical Learning](#), 2nd Edition, section 9.2.4

Categorical features: multi-class classification

- ❑ In Spark, for multiclass classification, all $2^{M-1} - 1$ possible splits are used whenever possible.
- ❑ When $2^{M-1} - 1$ is greater than the `maxBins` parameter, Spark uses a (heuristic) method similar to the method used for binary classification and regression.
- ❑ The M categorical feature values are ordered by impurity, and the resulting $M - 1$ split candidates are considered.

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

APIs for decision trees

- ❑ The class for classification is `DecisionTreeClassifier`.
- ❑ The class for regression is `DecisionTreeRegressor`.

The DecisionTreeClassifier class

```
class sklearn.tree. DecisionTreeClassifier(criterion='gini', splitter='best',
max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_features=None, random_state=None, max_leaf_nodes=None,
min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

[\[source\]](#)

DecisionTreeClassifier class in scikit-learn

```
class pyspark.ml.classification. DecisionTreeClassifier(featuresCol='features',
labelCol='label', predictionCol='prediction', probabilityCol='probability',
rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,
impurity='gini', seed=None)
```

[\[source\]](#)

DecisionTreeClassifier class in pyspark

The DecisionTreeRegressor class

```
class sklearn.tree.DecisionTreeRegressor(*, criterion='mse', splitter='best', max_depth=None,  
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None,  
random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None,  
ccp_alpha=0.0)
```

[\[source\]](#)

DecisionTreeRegressor class in scikit-learn

```
class pyspark.ml.regression.DecisionTreeRegressor(featuresCol='features',  
labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,  
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,  
impurity='variance', seed=None, varianceCol=None, weightCol=None, leafCol='',  
minWeightFractionPerNode=0.0)
```

[\[source\]](#)

DecisionTreeRegressor class in pyspark

Parameters to adjust

- **maxDepth**: Maximum depth of a tree.
- **maxBins**: Max number of bins for discretizing continuous features.
Must be ≥ 2 and \geq number of categories for any categorical feature.
- **impurity**: Criterion used for information gain calculation
(case-insensitive). Supported options: `entropy` or `gini` for
classification and `variance` for regression.
- There are several other parameters some of which will be reviewed in
the Lab session.

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

PLANET: horizontal partitioning

- PLANET is the standard algorithm to train a decision tree in a distributed dataset or horizontal partitioning.
- Let $\mathbf{X} \in \mathbb{R}^{n \times p}$ be the input matrix

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1^T \\ \vdots \\ \mathbf{x}_n^T \end{bmatrix},$$

where $\mathbf{x}_k \in \mathbb{R}^{p \times 1}$, meaning that there are p features.

- Horizontal partitioning refers to the fact that each worker will receive a subset of the n instances or row vectors.

Setup

- The following description is in the paper [Yggdrasil: An Optimized System for Training Deep Decision Trees at Scale](#)
- Assume there are B potential thresholds for each of the p features considered.
- Let us define S as the set of cardinality $p \times B$ that contains all the split candidates.
- Assume there are k workers. Each worker j computes c sufficient statistics over a subset of the original data, \mathbf{X}_j .
- What are these sufficient statistics?
 - classification: label counts (e.g. $c = 4$ for binary classification, 2 per branch).
 - regression: count, sum, and sum² (so, $c = 6$, 3 per branch).

Setup

- ❑ Each worker j then computes the sufficient statistics $g_j(s)$ over \mathbf{X}_j for each $s \in \mathcal{S}$.
- ❑ For a parent node i in the tree, the optimal split s^* is found by solving

$$s^* = \arg \max_{s \in \mathcal{S}} f \left(\sum_{j=1}^k g_j(s) \right),$$

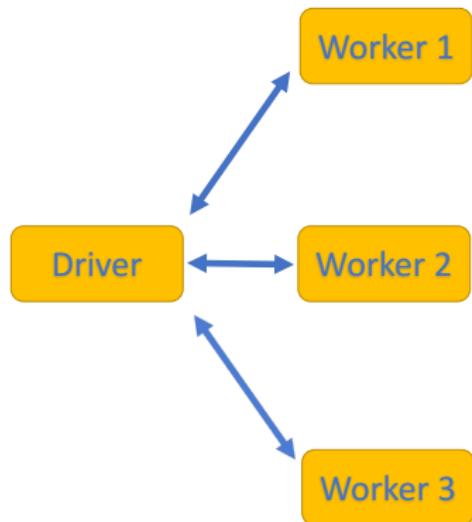
where $f : \mathbb{R}^c \rightarrow \mathbb{R}$ computes the information gains.

- ❑ Notice also that the function $g_j(s)$ computed using the subset of the data \mathbf{X}_j can be computed using a map-reduce approach at the level of the worker.
- ❑ The description above leads to a natural distributed algorithm.

Algorithm to compute s^* in a distributed fashion

- We start by assuming the depth of the tree is fixed to D .
- At iteration t ,
 - the algorithm computes the optimal splits for **all** the nodes on the level t of the tree.
 - there is a single round trip of communication between the master and the workers.
- Each tree node i is split as follows
 1. The j -th worker locally computes sufficient statistics $g_j(s)$ for all $s \in S$.
 2. Each worker communicates all statistics $g_j(s)$ to the master (Bp in total).
 3. The master computes the best split s^* .
 4. The master broadcasts s^* to the workers, who update their local states to keep track of which instances are assigned to which child nodes.

The distributed dataset



- Spam

 Get viagra real cheap! Send money to ...
 - Ham

 Dear Professor, I have a question about Spark ...
 - Ham

 Here is the info I was supposed to send you ...
-
- Spam

 Your bank account has been hacked! Go to ...
 - Ham

 Hi! I completed the Scala assignment ...
 - Ham

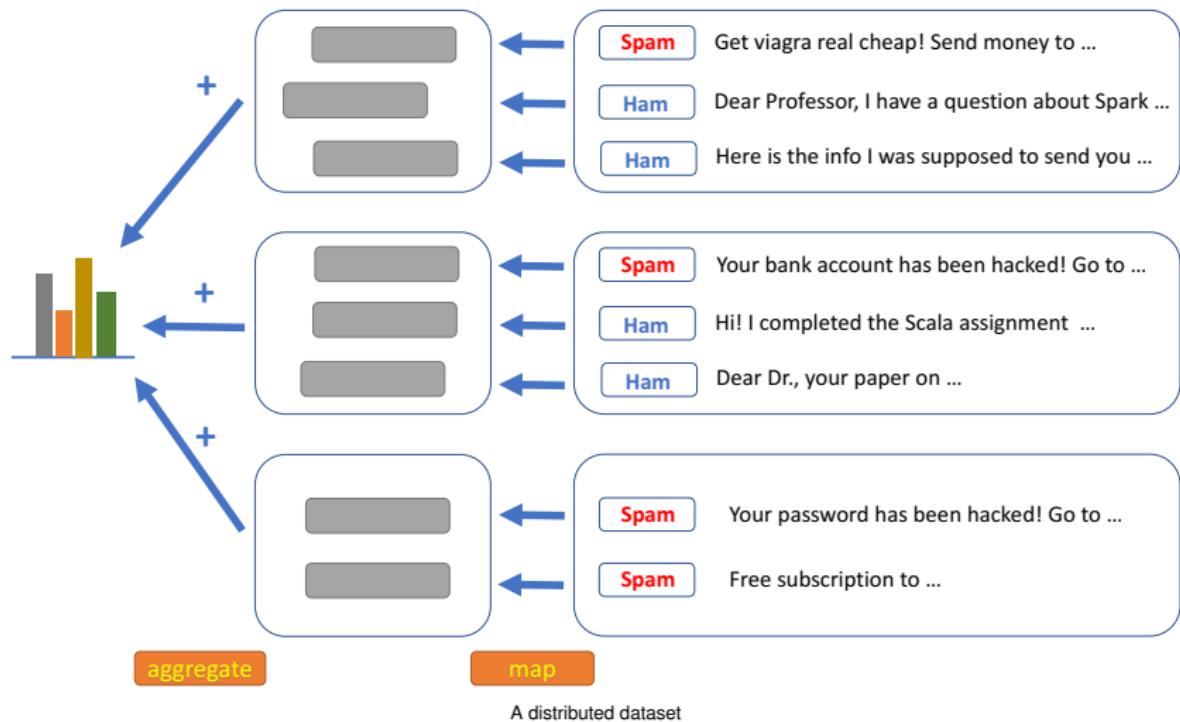
 Dear Dr., your paper on ...
-
- Spam

 Your password has been hacked! Go to ...
 - Spam

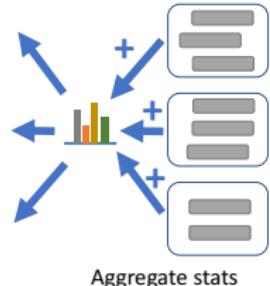
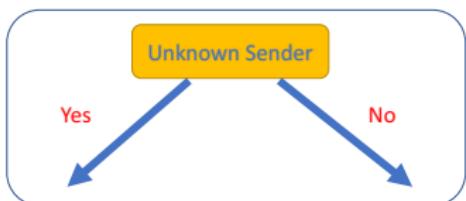
 Free subscription to ...

A distributed dataset

The distributed dataset

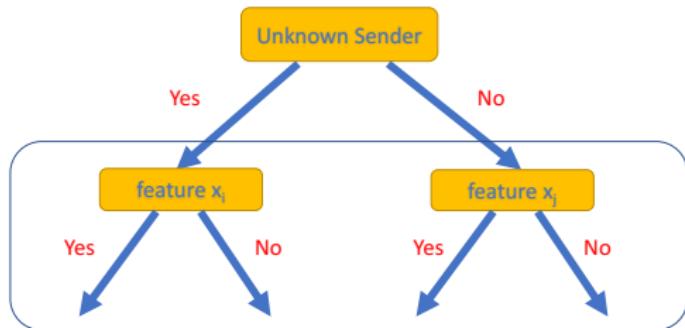


Building a decision tree

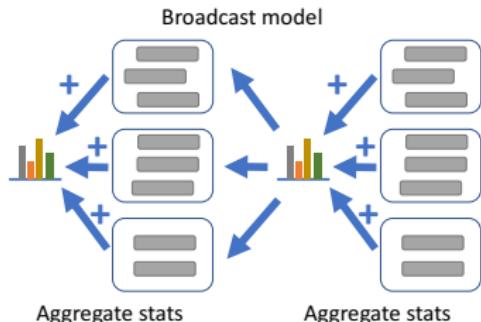


Building the tree in Spark

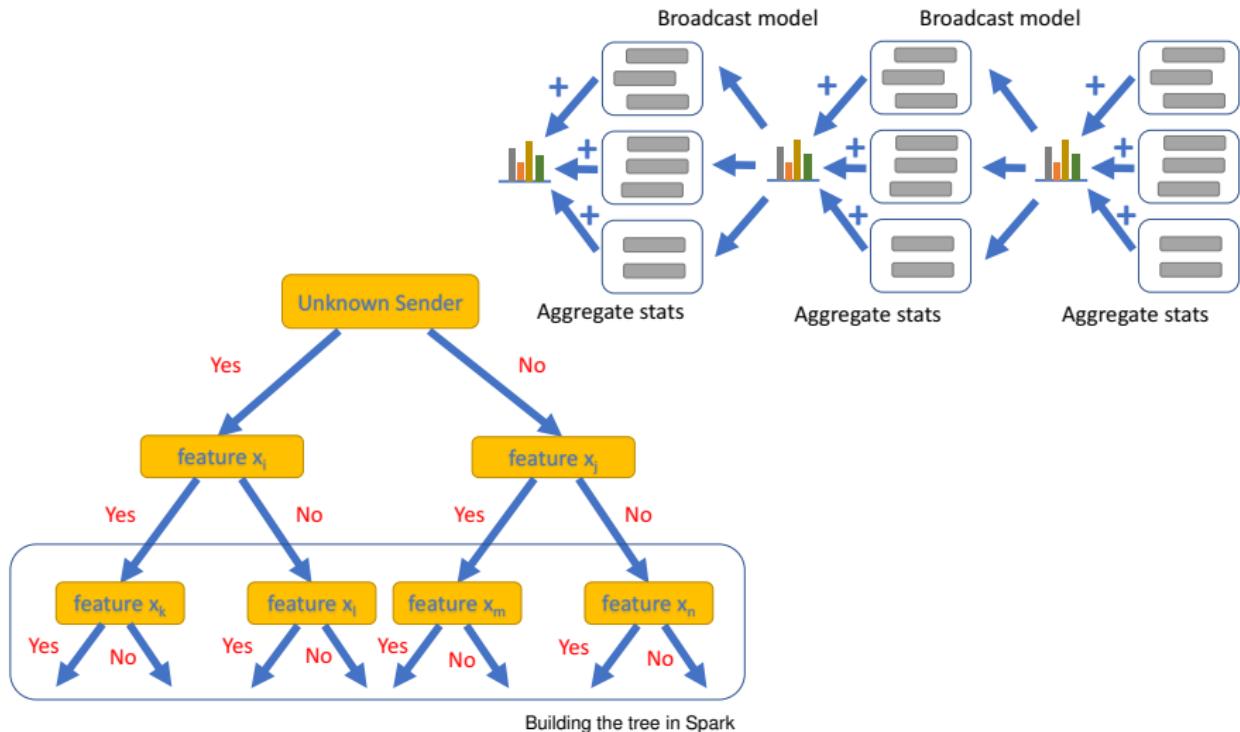
Building a decision tree



Building the tree in Spark



Building a decision tree



Computational and communication complexity

- Computation is linear in n , p , and D , so easy to parallelize.
- The issue is the communication overhead.
- For each tree node, step 2 in the algorithm before, requires communicating kBp tuples of size c .
- With 2^D total nodes, the total communication is $2^D kBpc$ floating point values.
- Communication is then exponential in tree depth D and linear in thresholds B .

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

Bagging and boosting

- Rather than creating a single model they generate a set of models and then make predictions by aggregating the outputs of these models.
- A prediction model that is composed of a set of models is called a **model ensemble**.
- In order for this approach to work the models that are in the ensemble must be different from each other.
- There are two standard approaches to creating ensembles: bagging and boosting.

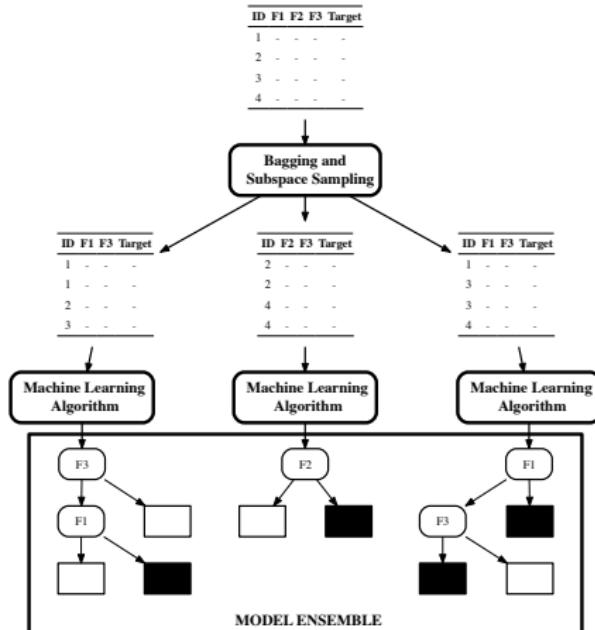
Bagging: Definition

- When we use **bagging** (or **bootstrap aggregating**) each model in the ensemble is trained on a random sample of the dataset known as **bootstrap samples**.
- Each random sample is the same size as the dataset and **sampling with replacement** is used.
- Consequently, every bootstrap sample will be missing some of the instances from the dataset so each bootstrap sample will be different and this means that models trained on different bootstrap samples will also be different

Bagging: Random forest

- When bagging is used with decision trees each bootstrap sample only uses a randomly selected subset of the descriptive features in the dataset. This is known as **subspace sampling**.
- The combination of bagging, subspace sampling, and decision trees is known as a **random forest** model.

Example of using bagging



The process of creating a model ensemble using bagging and subspace sampling.

Boosting: How does it work?

- ❑ Boosting works by iteratively creating models and adding them to the ensemble.
- ❑ The iteration stops when a predefined number of models have been added.
- ❑ When we use **boosting** each new model added to the ensemble is biased to pay more attention to instances that previous models miss-classified.
- ❑ This is done by incrementally adapting the dataset used to train the models. To do this we use a **weighted dataset**

Boosting: Weighted Dataset

- Each instance has an associated weight $\mathbf{w}_i \geq 0$,
- Initially set to $\frac{1}{n}$ where n is the number of instances in the dataset.
- After each model is added to the ensemble it is tested on the **training data** and the weights of the instances the model gets correct are decreased and the weights of the instances the model gets incorrect are increased.
- These weights are used as a distribution over which the dataset is sampled to create a **replicated training set**, where the replication of an instance is proportional to its weight.

Algorithm

During each **training iteration** the algorithm:

1. Induces a model and calculates the total error, ϵ , by summing the weights of the training instances for which the predictions made by the model are incorrect.
2. Increases the weights for the instances misclassified using:

$$\mathbf{w}[i] \leftarrow \mathbf{w}[i] \times \left(\frac{1}{2 \times \epsilon} \right)$$

3. Decreases the weights for the instances correctly classified:

$$\mathbf{w}[i] \leftarrow \mathbf{w}[i] \times \left(\frac{1}{2 \times (1 - \epsilon)} \right)$$

4. Calculate a **confidence factor**, α , for the model such that α increases as ϵ decreases:

$$\alpha = \frac{1}{2} \times \log_e \left(\frac{1 - \epsilon}{\epsilon} \right)$$

Predictions

- Once the set of models have been created the ensemble makes **predictions** using a weighted aggregate of the predictions made by the individual models.
- The weights used in this aggregation are simply the confidence factors associated with each model.

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

Review of ensemble methods

Ensemble methods in PySpark

References

Bagging and Boosting in PySpark

- ❑ Random Forests are implemented in PySpark.
- ❑ The boosting model implemented in PySpark is the Gradient-Boosted Trees (GBTs).

Random forests

- ❑ spark.ml supports random forests for binary and multiclass classification and for regression, using both continuous and categorical features.

- ❑ spark.ml implements random forests using the existing decision tree implementation.

API Random forests

```
class pyspark.ml.classification.RandomForestClassifier(featuresCol='features',
labelCol='label', predictionCol='prediction', probabilityCol='probability',
rawPredictionCol='rawPrediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,
impurity='gini', numTrees=20, featureSubsetStrategy='auto', seed=None, subsamplingRate=1.0,
leafCol='', minWeightFractionPerNode=0.0, weightCol=None, bootstrap=True) [source]
```

RandomForestClassifier class in pyspark

```
class pyspark.ml.regression.RandomForestRegressor(featuresCol='features',
labelCol='label', predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1,
minInfoGain=0.0, maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10,
impurity='variance', subsamplingRate=1.0, seed=None, numTrees=20,
featureSubsetStrategy='auto', leafCol='', minWeightFractionPerNode=0.0, weightCol=None,
bootstrap=True) [source]
```

RandomForestRegressor class in pyspark

Two important parameters to adjust

- ❑ **numTrees**: Number of trees in the forest.
- ❑ **maxDepth**: Maximum depth of each tree in the forest.
- ❑ There are several other parameters some of which will be reviewed in the Lab session.

Gradient-boosted trees

- ❑ `spark.ml` supports GBTs for binary classification and for regression, using both continuous and categorical features.
- ❑ `spark.ml` implements GBTs using the existing decision tree implementation.
- ❑ GBTs do not yet support multiclass classification.

API Gradient-boosted trees

```
class pyspark.ml.classification.GBTClassifier(featuresCol='features', labelCol='label',  
predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0,  
maxMemoryInMB=256, cacheNodeIds=False, checkpointInterval=10, lossType='logistic',  
maxIter=20, stepSize=0.1, seed=None, subsamplingRate=1.0, impurity='variance',  
featureSubsetStrategy='all', validationTol=0.01, validationIndicatorCol=None, leafCol='',  
minWeightFractionPerNode=0.0, weightCol=None)
```

[\[source\]](#)

GBTClassifier class in pyspark

```
class pyspark.ml.regression.GBTRegressor(featuresCol='features', labelCol='label',  
predictionCol='prediction', maxDepth=5, maxBins=32, minInstancesPerNode=1, minInfoGain=0.0,  
maxMemoryInMB=256, cacheNodeIds=False, subsamplingRate=1.0, checkpointInterval=10,  
lossType='squared', maxIter=20, stepSize=0.1, seed=None, impurity='variance',  
featureSubsetStrategy='all', validationTol=0.01, validationIndicatorCol=None, leafCol='',  
minWeightFractionPerNode=0.0, weightCol=None)
```

[\[source\]](#)

GBTRegressor class in pyspark

Parameters to adjust

- The parameters to adjust are similar to the ones used for random forests.
- They will be reviewed in the Lab session.

Contents

Review of decision trees

How do decision trees work in Spark?

Building the trees and impurity measures in Spark

Split candidates

Decision tree classes in `spark.ml`

PLANET algorithm

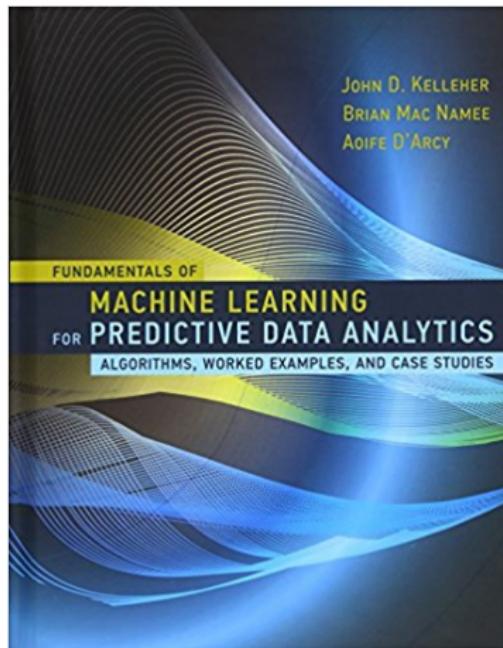
Review of ensemble methods

Ensemble methods in PySpark

References

References used in this lecture

Book: *Fundamentals of Machine Learning for Predictive Analytics* by Kelleher, Mac Namee and D'Arcy, 2015.



References used in this lecture

Website: *Spark Python API Documentation.*

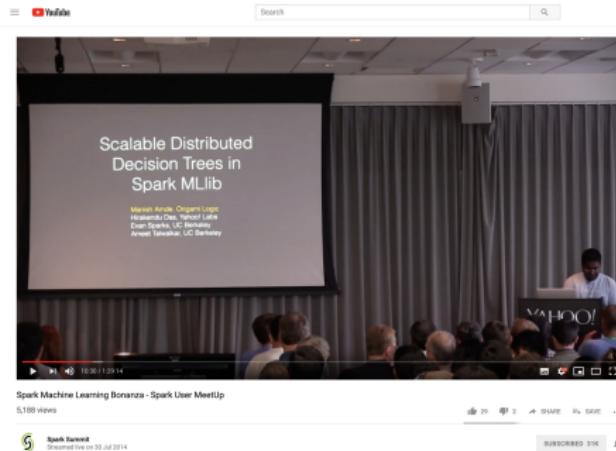
The screenshot shows the Apache Spark API Reference page. At the top, there's a navigation bar with links to 'Getting Started', 'User Guide', 'API Reference' (which is the active page), 'Development', and 'Migration Guide'. Below the navigation bar is a search bar labeled 'Search the docs ...'. To the left, there's a sidebar with a 'Spark' logo and a list of categories: 'Spark SQL', 'Pandas API on Spark', 'Structured Streaming', 'MLlib (DataFrame-based)', 'Spark Streaming', 'MLlib (RDD-based)', 'Spark Core', and 'Resource Management'. The main content area is titled 'API Reference' and contains a sub-header: 'This page lists an overview of all public PySpark modules, classes, functions and methods.' Below this, there's a hierarchical list of API sections:

- **Spark SQL**
 - Core Classes
 - Spark Session APIs
 - Configuration
 - Input and Output
 - DataFrame APIs
 - Column APIs
 - Data Types
 - Row
 - Functions
 - Window
 - Grouping
 - Catalog APIs
- **Pandas API on Spark**
 - Input/Output
 - General functions
 - Series
 - DataFrame
 - Index objects
 - Window
 - GroupBy
 - Machine Learning utilities
 - Extensions
- **Structured Streaming**
 - Core Classes
 - Input and Output
 - Query Management
- **MLlib (DataFrame-based)**
 - Pipeline APIs
 - Parameters
 - Feature

<https://spark.apache.org/docs/latest/api/python/reference/index.html>

References used in this lecture

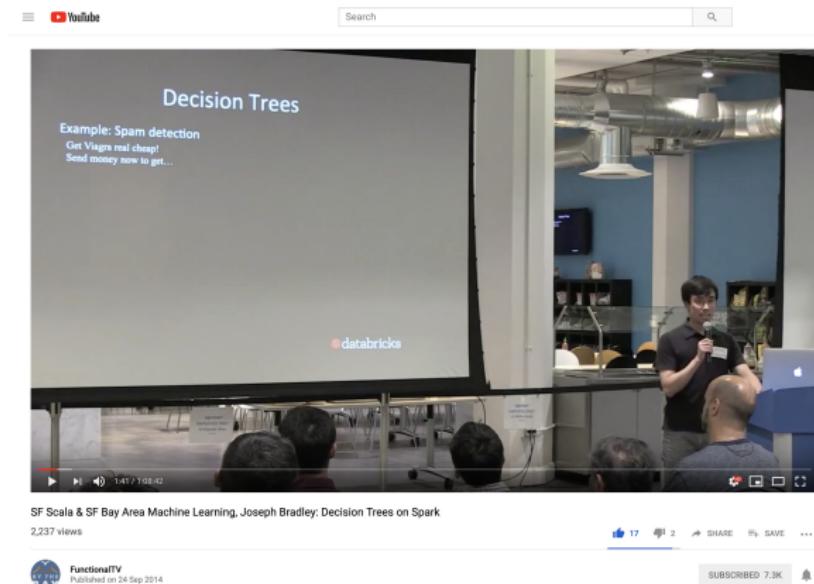
Youtube video: *Scalable Decision Trees in Spark MLlib* by Manish Amde
(contributor to the MLlib implementation of Decision Trees in Spark).



<https://www.youtube.com/watch?v=N453EV5gHRA&t=10m30s>

References used in this lecture

Youtube video: *Decision Trees on Spark* by Joseph Bradley (from databricks).



<https://www.youtube.com/watch?v=3WS9OK3EXVA>

References used in this lecture

Paper: *PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce* by B. Panda et al. (from Google).

PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Biswanath Panda, Joshua S. Herbach, Sugato Basu, Roberto J. Bayardo
Google, Inc.

[bpanda, jsherbach, sugato]@google.com, bayardo@alum.mit.edu

ABSTRACT

Classification and regression tree learning on massive datasets is a common data mining task at Google, yet many state of the art tree learning algorithms require training data to reside in memory on a single machine. While more scalable implementations of tree learning have been proposed, they typically require specialized parallel computing architectures. In contrast, the majority of Google's computing infrastructure is based on commodity hardware.

In this paper, we describe PLANET: a scalable distributed framework for learning tree models over large datasets. PLANET defines tree learning as a series of distributed computations, and implements each one using the *MapReduce* model of distributed computation. We show how this framework supports scalable construction of classification and regression trees, as well as ensembles of such models. We discuss the benefits and challenges of using a MapReduce compute cluster for tree learning, and demonstrate the scalability of this approach by applying it to a real world learning task from the domain of computational advertising.

plexities such as data partitioning, scheduling tasks across many machines, handling machine failures, and performing inter-machine communication. These properties have motivated many technology companies to run MapReduce frameworks on their compute clusters for data analysis and other data management tasks. MapReduce has become in some sense an industry standard. For example, there are open source implementations such as Hadoop that can be run either in-house or on cloud computing services such as Amazon EC2.¹ Startups like Cloudera² offer software and services to simplify Hadoop deployment, and companies including Google, IBM and Yahoo! have granted several universities access to Hadoop clusters to further cluster computing research.³

Despite the growing popularity of MapReduce [12], its application to certain standard data mining and machine learning tasks remains poorly understood. In this paper we focus on one such task: tree learning. We believe that a tree learner capable of exploiting a MapReduce cluster can effectively address many scalability issues that arise in building tree models on massive datasets. Our choice of focusing

Scalable logistic regression

Mauricio A. Álvarez, PhD

Scalable Machine Learning,
University of Sheffield



The
University
Of
Sheffield.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Probabilistic classifier

- A logistic regression model is an example of a probabilistic classifier.
- Let $\mathbf{x} \in \mathbb{R}^p$ represents a feature vector and y the target value.
- For a binary classification problem we can use $y \in \{0, 1\}$ or $y \in \{-1, +1\}$.
- We model the relationship between y , and \mathbf{x} using a Bernoulli distribution.

Bernoulli distribution (I)

- A Bernoulli random variable Y is a random variable that can only take two possible values.
- For example, the random variable Y associated to the experiment of tossing a coin.
- Output “heads” is assigned 1 ($Y = 1$), and output “tails” is assigned 0 ($Y = 0$).

Bernoulli distribution (II)

- A Bernoulli distribution is a probability distribution for Y , expressed as

$$p(Y = y) = \text{Ber}(y|\mu) = \begin{cases} \mu & y = 1, \\ 1 - \mu & y = 0, \end{cases}$$

where $\mu = P(Y = 1)$.

- The expression above can be summarized in one line using

$$p(Y = y) = \text{Ber}(y|\mu) = \mu^y(1 - \mu)^{1-y},$$

How are y and \mathbf{x} related in logistic regression?

- The target feature y follows a Bernoulli distribution

$$p(y|\mathbf{x}) = \text{Ber}(y|\mu(\mathbf{x})).$$

- Notice how the probability $\mu = P(y = 1)$ explicitly depends on \mathbf{x} .
- In logistic regression, the probability $\mu(\mathbf{x})$ is given as

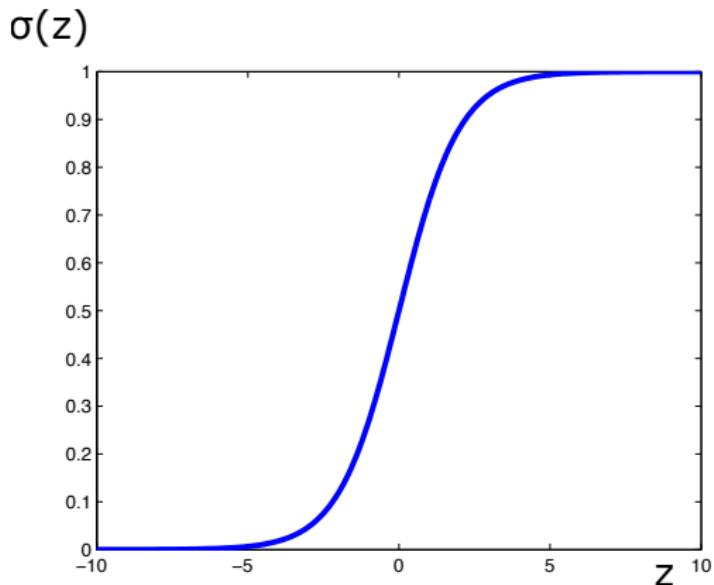
$$\mu(\mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})} = \sigma(\mathbf{w}^\top \mathbf{x}),$$

where $\sigma(z)$ is known as the *logistic sigmoid* function.

- We then have

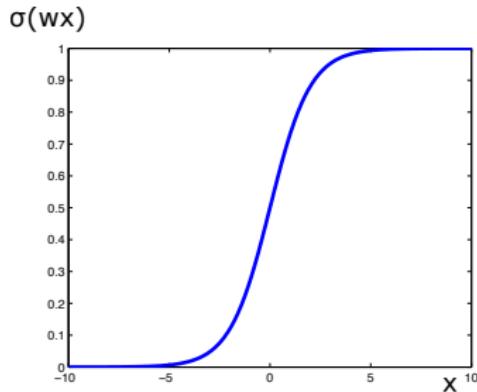
$$p(y|\mathbf{w}, \mathbf{x}) = \text{Ber}(y|\sigma(\mathbf{w}^\top \mathbf{x})).$$

The logistic sigmoid function $\sigma(z)$



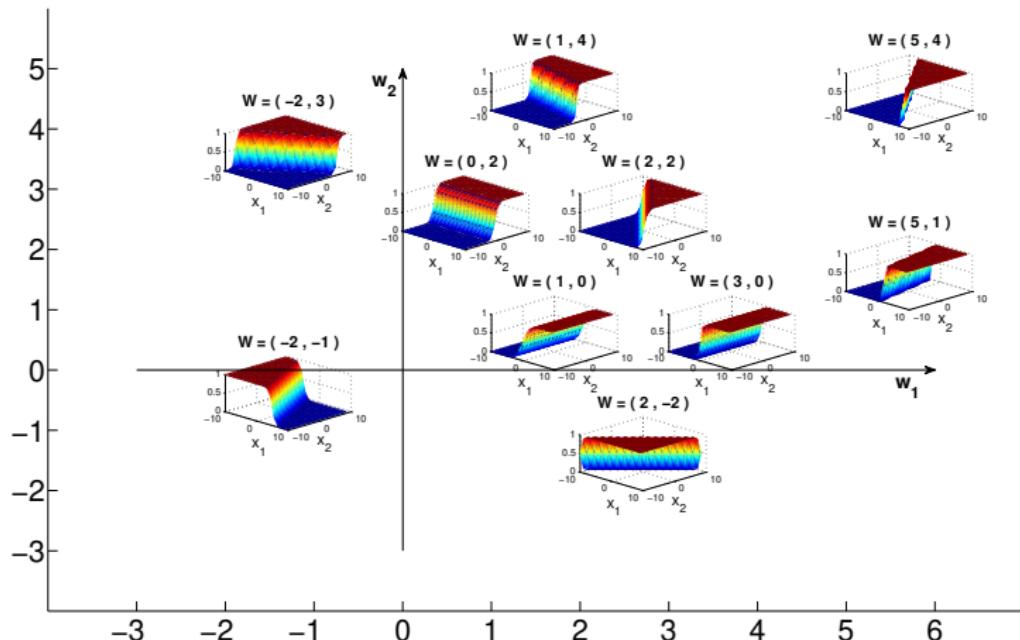
- Recall $\sigma(z) = \frac{1}{1 + \exp(-z)}$.
- If $z \rightarrow \infty$, $\sigma(z) = 1$. If $z \rightarrow -\infty$, $\sigma(z) = 0$. If $z = 0$, $\sigma(z) = 0.5$.

The logistic sigmoid function $\sigma(\mathbf{w}^\top \mathbf{x})$



- We have $z = \mathbf{w}^\top \mathbf{x}$. For simplicity, assume $\mathbf{x} = x$, then $\sigma(wx)$.
- Recall $\sigma(wx) = \frac{1}{1 + \exp(-wx)}$.
- So $\frac{d\sigma(wx)}{dx} \Big|_{x=0} = \frac{w}{4}$.

The logistic sigmoid function in 2d



Plot of $\sigma(w_1x_1 + w_2x_2)$. Here $\mathbf{w} = [w_1 \ w_2]^\top$.

Decision boundary

- After the training phase, we will have an estimator for \mathbf{w} .
- For a test input vector \mathbf{x}_* , we compute $p(y = 1 | \mathbf{w}, \mathbf{x}_*) = \sigma(\mathbf{w}^\top \mathbf{x}_*)$.
- This will give us a value between 0 and 1.
- We define a threshold of 0.5 to decide to which class we assign \mathbf{x}_* .
- With this threshold we induce a linear decision boundary in the input space.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Cross-entropy error function

- Let $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$.
- We write $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_N]^\top$, and $\mathbf{y} = [y_1 \cdots y_N]^\top$.
- Assuming IID observations

$$p(\mathbf{y}|\mathbf{w}, \mathbf{X}) = \prod_{n=1}^N p(y_n|\mathbf{w}, \mathbf{x}_n) = \prod_{n=1}^N \text{Ber}(y_n|\sigma(\mathbf{w}^\top \mathbf{x}_n)).$$

- The cross-entropy function or negative log-likelihood is given as

$$\begin{aligned} NLL(\mathbf{w}) &= -\log p(\mathbf{y}|\mathbf{w}, \mathbf{X}) \\ &= -\sum_{n=1}^N \{y_n \log[\sigma(\mathbf{w}^\top \mathbf{x}_n)] + (1 - y_n) \log[1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)]\}, \end{aligned}$$

which can be minimised with respect to \mathbf{w} .

Gradient and Hessian of $NLL(\mathbf{w})$

- It can be shown that the gradient $\mathbf{g}(\mathbf{w})$ of $NLL(\mathbf{w})$ is given as

$$\mathbf{g}(\mathbf{w}) = \frac{d}{d\mathbf{w}} NLL(\mathbf{w}) = \sum_{n=1}^N [\sigma(\mathbf{w}^\top \mathbf{x}_n) - y_n] \mathbf{x}_n = \mathbf{X}^\top (\boldsymbol{\sigma} - \mathbf{y}),$$

where $\boldsymbol{\sigma} = [\sigma(\mathbf{w}^\top \mathbf{x}_1) \cdots \sigma(\mathbf{w}^\top \mathbf{x}_N)]^\top$.

- It can also be shown that the Hessian $\mathbf{H}(\mathbf{w})$ of $NLL(\mathbf{w})$ is given as

$$\mathbf{H}(\mathbf{w}) = \frac{d}{d\mathbf{w}} \mathbf{g}(\mathbf{w})^\top = \sum_{n=1}^N \sigma(\mathbf{w}^\top \mathbf{x}_n) [1 - \sigma(\mathbf{w}^\top \mathbf{x}_n)] \mathbf{x}_n \mathbf{x}_n^\top = \mathbf{X}^\top \boldsymbol{\Sigma} \mathbf{X},$$

where $\boldsymbol{\Sigma} = \text{diag}([\sigma(\mathbf{w}^\top \mathbf{x}_1)[1 - \sigma(\mathbf{w}^\top \mathbf{x}_1)], \dots, \sigma(\mathbf{w}^\top \mathbf{x}_N)[1 - \sigma(\mathbf{w}^\top \mathbf{x}_N)]])$.

Contents

Logistic regression

Model fitting

Cross-entropy error function

Optimisation routines

Efficient computation of the Gradient and the Hessian

Batch gradient descent

Stochastic Gradient Descent

Beyond basic logistic regression

Regularisation

Multi-class logistic regression

Logistic regression in PySpark

References

General problem

- We are given a function $f(\mathbf{w})$, where $\mathbf{w} \in \mathbb{R}^p$.
- Aim: to find a value for \mathbf{w} that minimises $f(\mathbf{w})$.
- Use an iterative procedure

$$\mathbf{w}_{k+1} = \mathbf{w}_k + \eta \mathbf{d}_k,$$

where \mathbf{d}_k is known as the search direction and it is such that

$$f(\mathbf{w}_{k+1}) < f(\mathbf{w}_k).$$

- The parameter η is known as the **step size** or **learning rate**.

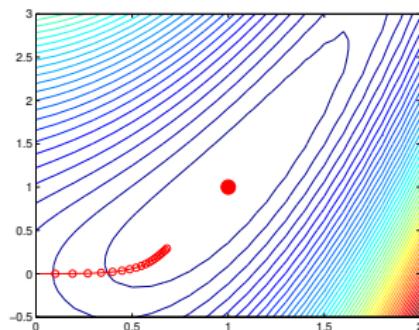
Gradient descent

- Perhaps, the simplest algorithm for unconstrained optimisation.
- It assumes that $\mathbf{d}_k = -\mathbf{g}_k$, where $\mathbf{g}_k = \mathbf{g}(\mathbf{w}_k)$.
- Also known as **steepest descent**.
- It can be written like

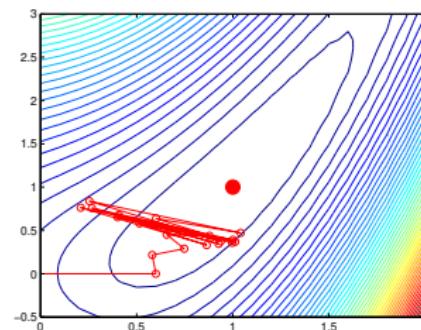
$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{g}_k.$$

Step size

- ❑ The main issue in gradient descent is how to set the step size.
- ❑ If it is too small, convergence will be very slow. If it is too large, the method can fail to converge at all.



(a)



(b)

The function to optimise is $f(w_1, w_2) = 0.5(w_1^2 - w_2)^2 + 0.5(w_1 - 1)^2$. The minimum is at $(1, 1)$. In (a) $\eta = 0.1$. In (b) $\eta = 0.6$.

Alternatives to choose the step size η

- Line search methods (there are different alternatives).
- Line search methods may use search directions other than the steepest descent direction.
- Conjugate gradient (method of choice for quadratic objectives $f(\mathbf{w}) = \mathbf{w}^\top \mathbf{A}\mathbf{w}$).

Newton's method (I)

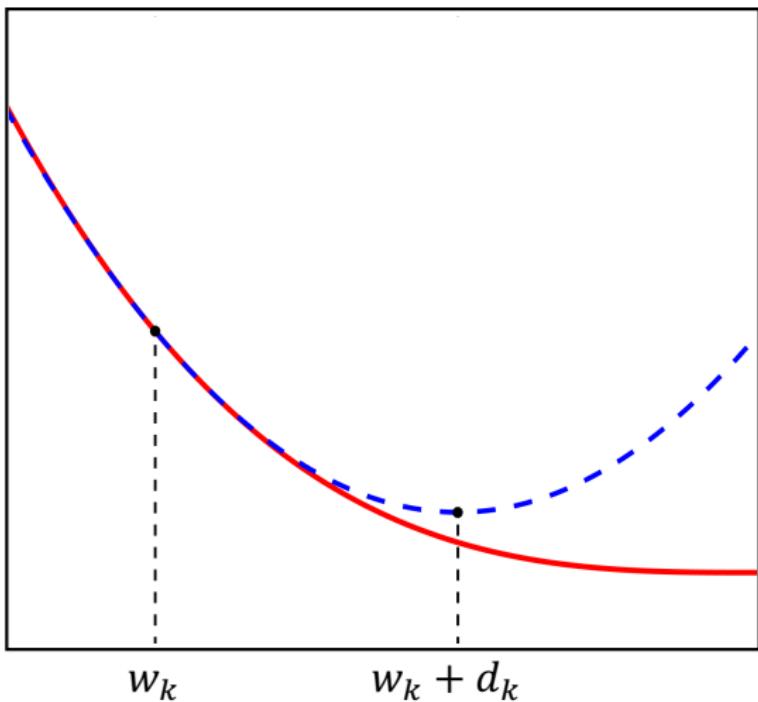
- Another important search direction is the *Newton* direction.
- It derives a faster optimisation algorithm by taking the curvature of the space (i.e., the Hessian) into account.
- Optimisation methods that include the Hessian are also known as second order optimisation methods.
- In Newton's algorithm

$$\mathbf{w}_{k+1} = \mathbf{w}_k - \eta \mathbf{H}_k^{-1} \mathbf{g}_k,$$

where $\mathbf{g}_k = \mathbf{g}(\mathbf{w}_k)$, and $\mathbf{H}_k = \mathbf{H}(\mathbf{w}_k)$.

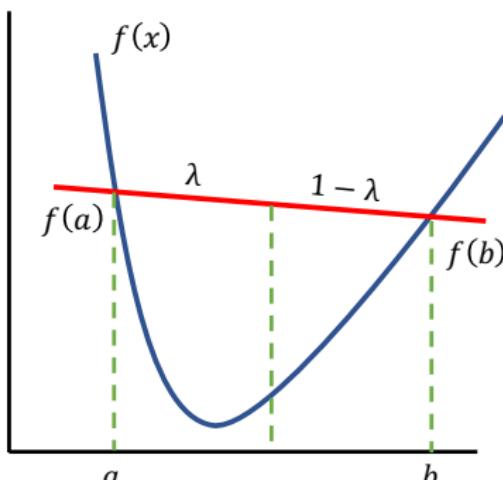
- Usually $\eta = 1$, and $\mathbf{d}_k = -\mathbf{H}_k^{-1} \mathbf{g}_k$.

Newton's method (II)

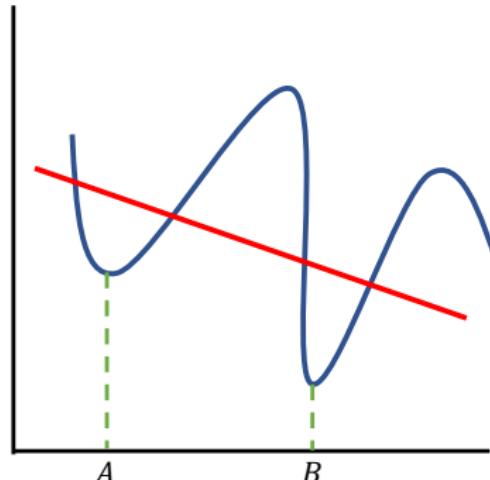


Newton's method and convex functions (I)

- Newton's method requires that \mathbf{H}_k be positive definite (p.d.)
- The condition holds if the function is strictly convex.



(a)



(b)

In (a), an illustration of a convex function. The chord joining $(a, f(a))$ and $(b, f(b))$ lies above the function. In (b), a function that is neither convex nor concave.

Newton's method and convex functions (II)

- If not strictly convex, \mathbf{H}_k may not be p.d., and $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$ may not be a descent direction.
- Alternatives:
 - If \mathbf{H}_k is not positive definite, use $\mathbf{d}_k = -\mathbf{g}_k$ instead.
 - Use the *Levenberg Marquardt* algorithm, which compromises between the Newton direction and the steepest direction.
 - Rather than computing $\mathbf{d}_k = -\mathbf{H}_k^{-1}\mathbf{g}_k$, find \mathbf{d}_k that solves the linear system $\mathbf{H}_k\mathbf{d}_k = -\mathbf{g}_k$ using an iterative procedure. Stop when needed (**truncated Newton**).

Quasi-Newton methods

- The main drawback of the Newton direction is the need for the Hessian.
- Explicit computation of this matrix of second derivatives can sometimes be a cumbersome, error-prone, and expensive process.
- Quasi-Newton search directions provide an attractive alternative to Newton's method.
- They do not require computation of the Hessian and yet still attain a faster rate of convergence.
- In place of the true Hessian \mathbf{H}_k , they use an approximation \mathbf{B}_k , which is updated after each step to take account of the additional knowledge gained during the step.

BFGS formula for \mathbf{H}_k

- In the Broyden, Fletcher, Goldfarb, and Shanno (BFGS) formula

$$\mathbf{B}_{k+1} = \mathbf{B}_k + \frac{\mathbf{z}_k \mathbf{z}_k^\top}{\mathbf{z}_k^\top \mathbf{s}_k} - \frac{(\mathbf{B}_k \mathbf{s}_k)(\mathbf{B}_k \mathbf{s}_k)^\top}{\mathbf{s}_k^\top \mathbf{B}_k \mathbf{s}_k},$$

where $\mathbf{s}_k = \mathbf{w}_k - \mathbf{w}_{k-1}$ and $\mathbf{z}_k = \mathbf{g}_k - \mathbf{g}_{k-1}$.

- This is a rank-two update to the matrix, and ensures that the matrix remains positive definite.
- Usually $\mathbf{B}_0 = \mathbf{I}$.
- The search direction is then $\mathbf{d}_k = -\mathbf{B}_k^{-1} \mathbf{g}_k$.

BFGS formula for \mathbf{H}_k^{-1}

- BFGS can alternatively update $\mathbf{C}_k = \mathbf{H}_k^{-1}$ using

$$\mathbf{C}_{k+1} = \left(\mathbf{I} - \frac{\mathbf{s}_k \mathbf{z}_k^\top}{\mathbf{z}_k^\top \mathbf{s}_k} \right) \mathbf{C}_k \left(\mathbf{I} - \frac{\mathbf{z}_k \mathbf{s}_k^\top}{\mathbf{z}_k^\top \mathbf{s}_k} \right) + \frac{\mathbf{s}_k \mathbf{s}_k^\top}{\mathbf{z}_k^\top \mathbf{s}_k}.$$

- The search direction is then $\mathbf{d}_k = -\mathbf{C}_k \mathbf{g}_k$.

Limited memory BFGS (L-BFGS)

- Since storing the Hessian takes $O(p^2)$ space, for very large problems, one can use limited memory BFGS, or L-BFGS.
- In L-BFGS, \mathbf{H}_k or \mathbf{H}_k^{-1} is approximated by a diagonal plus low rank matrix.
- In particular, the product $\mathbf{B}_k^{-1} \mathbf{g}_k$ can be obtained by performing a sequence of inner products with \mathbf{s}_k and \mathbf{z}_k , using only the m most recent $(\mathbf{s}_k, \mathbf{z}_k)$ pairs, and ignoring older information.
- The storage requirements are therefore $O(mp)$. Typically $m \sim 20$ suffices for good performance.

Optimisation methods applied to logistic regression

All the methods described above can be applied to find the parameter vector \mathbf{w} that minimises the negative log-likelihood $NLL(\mathbf{w})$ in logistic regression.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

Batch gradient descent

Stochastic Gradient Descent

Beyond basic logistic regression

Regularisation

Multi-class logistic regression

Logistic regression in PySpark

References

Algorithms in summation form

- When an algorithm sums over the data, we can easily distribute the calculations over multiple workers (e.g. cores).
- We just divide the dataset into as many pieces as there are workers, give each worker its share of the data to sum the equations over, and aggregate the result at the end.
- Both the gradient and Hessian in the optimisation algorithms we saw before have a summation form.
- Notice that the summation form does not change the underlying algorithm: it is not an approximation but an exact implementation.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

Batch gradient descent

Stochastic Gradient Descent

Beyond basic logistic regression

Regularisation

Multi-class logistic regression

Logistic regression in PySpark

References

Stochastic gradient descent (I)

- ❑ Traditionally in machine learning, the gradient \mathbf{g}_k and the Hessian \mathbf{H}_k are computed using the whole dataset $\mathcal{D} = \{\mathbf{x}_n, y_n\}_{n=1}^N$.
- ❑ There are settings, though, where only a subset of the data can be used.
- ❑ **Online learning:** the instances (\mathbf{x}_n, y_n) appear one at a time.
- ❑ **Large datasets:** computing the exact values for \mathbf{g}_k and \mathbf{H}_k would be expensive, if not impossible.

Stochastic gradient descent (II)

- ❑ In stochastic gradient descent (SGD), the gradient \mathbf{g}_k is computed using a subset of the instances available.
- ❑ The word stochastic refers to the fact that the value for \mathbf{g}_k will depend on the subset of the instances chosen for computation.

Stochastic gradient descent (III)

- In the stochastic setting, a better estimate can be found if the gradient is computed using

$$\mathbf{g}_k = \frac{1}{|S|} \sum_{i \in S} \mathbf{g}_{k,i},$$

where $S \in \mathcal{D}$, $|S|$ is the cardinality of S , and $\mathbf{g}_{k,i}$ is the gradient at iteration k computed using the instance (\mathbf{x}_i, y_i) .

- This setting is called Mini-batch gradient descent.
- For logistic regression, $\mathbf{g}_{k,i}$ would be the gradient computed for $-\log p(y_i | \mathbf{w}, \mathbf{x}_i)$.

Step size in SGD

- Choosing the value of η is particularly important in SGD since there is no easy way to compute it.
- Usually the value of η will depend on the iteration k , η_k .
- It should follow the **Robbins-Monro** conditions

$$\sum_{k=1}^{\infty} \eta_k = \infty, \quad \sum_{k=1}^{\infty} \eta_k^2 < \infty.$$

- Various formulas for η_k can be used

$$\eta_k = \frac{1}{k}, \quad \eta_k = \frac{1}{(\tau_0 + k)^\kappa},$$

where τ_0 slows down early iterations and $\kappa \in (0.5, 1]$.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

Regularisation

Multi-class logistic regression

Logistic regression in PySpark

References

What is regularisation?

- It refers to a technique used for preventing overfitting in a predictive model.
- It consists in adding a term (a regulariser) to the objective function that encourages simpler solutions.
- With regularisation, the objective function for logistic regression would be

$$f(\mathbf{w}) = NLL(\mathbf{w}) + \lambda R(\mathbf{w}),$$

where $R(\mathbf{w})$ is the regularisation term and λ the regularisation parameter.

- If $\lambda = 0$, we get $f(\mathbf{w}) = NLL(\mathbf{w})$.

Different types of regularisation

- The objective function for logistic regression would be

$$f(\mathbf{w}) = NLL(\mathbf{w}) + \lambda R(\mathbf{w}),$$

where $R(\mathbf{w})$ follows as

$$R(\mathbf{w}) = \alpha \|\mathbf{w}\|_1 + (1 - \alpha) \frac{1}{2} \|\mathbf{w}\|_2^2,$$

where $\|\mathbf{w}\|_1 = \sum_{m=1}^p |w_m|$, and $\|\mathbf{w}\|_2^2 = \sum_{m=1}^p w_m^2$.

- If $\alpha = 1$, we get ℓ_1 regularisation.
- If $\alpha = 0$, we get ℓ_2 regularisation.
- If $0 < \alpha < 1$, we get the elastic net regularisation.
- In Spark, λ is `regParam` and α is `elasticNetParam`.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Binary logistic regression for multi-class problems

- ❑ The implementation of multi-class logistic regression for K classes in Spark uses $K - 1$ binary logistic regression models.
- ❑ The idea is to choose one class as a “pivot” and train $K - 1$ binary logistic regression models between the pivot class and any of the other classes.

A pivot class

- ❑ In `spark.ml` the pivot class is the class “0”, and the probabilities for each class are given as

$$p(y = 0 | \mathbf{x}, \mathbf{w}) = \frac{1}{1 + \sum_{\ell=1}^{K-1} \exp(\mathbf{w}_\ell^\top \mathbf{x})}$$

$$p(y = k | \mathbf{x}, \mathbf{w}) = \frac{\exp(\mathbf{w}_k^\top \mathbf{x})}{1 + \sum_{\ell=1}^{K-1} \exp(\mathbf{w}_\ell^\top \mathbf{x})}, \quad k = 1, \dots, K - 1$$

- ❑ Given a dataset, it is possible to write an expression for the negative log-likelihood and then minimise it using any of the methods based on the gradient or the gradient and the Hessian.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

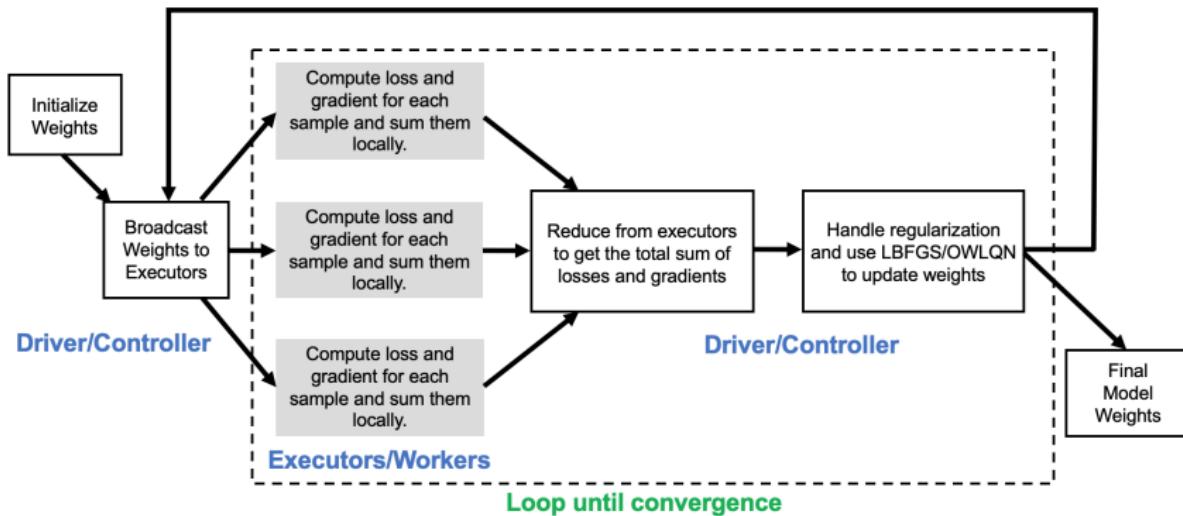
Beyond basic logistic regression

- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

Update of w in Apache Spark



LogisticRegression()

- ❑ LogisticRegression() works with DataFrames.
- ❑ It is possible to use regularisation ℓ_1 , ℓ_2 or Elastic Net.
- ❑ L-BFGS is used as a solver for LogisticRegression(), with ℓ_2 .
- ❑ When ℓ_1 , or Elastic Net are used, a variant of L-BFGS, known as Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) is used instead.

Parameters to adjust

- ❑ **maxIter**: max number of iterations.
- ❑ **regParam**: regularization parameter (≥ 0).
- ❑ **elasticNetParam**: the ElasticNet mixing parameter, in range [0, 1].
For $\alpha = 0$, the penalty is an ℓ_2 penalty. For $\alpha = 1$, it is an ℓ_1 penalty.
- ❑ **family**: the name of family which is a description of the label distribution to be used in the model. Supported options: auto, binomial, multinomial.
- ❑ **standardization**: whether to standardize the training features before fitting the model. It can be true or false.

Contents

Logistic regression

Model fitting

- Cross-entropy error function
- Optimisation routines

Efficient computation of the Gradient and the Hessian

- Batch gradient descent
- Stochastic Gradient Descent

Beyond basic logistic regression

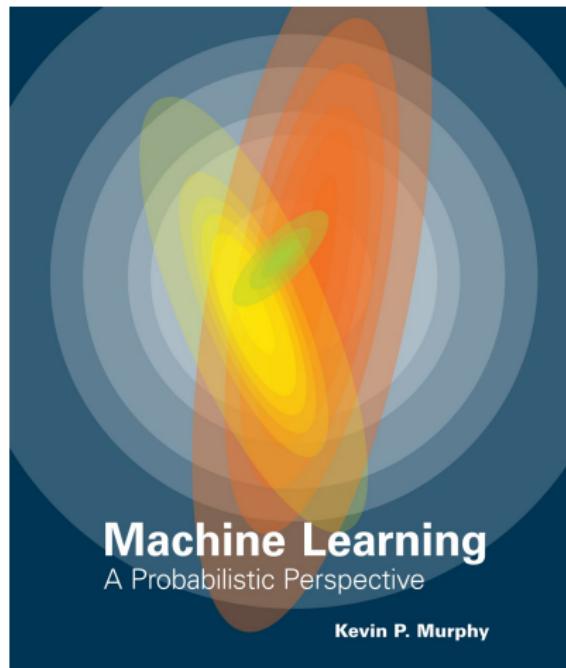
- Regularisation
- Multi-class logistic regression

Logistic regression in PySpark

References

References used in this lecture

Book: *Machine Learning: A Probabilistic Perspective* by Kevin P Murphy, 2012.



References used in this lecture

Website: *Spark Python API Documentation.*

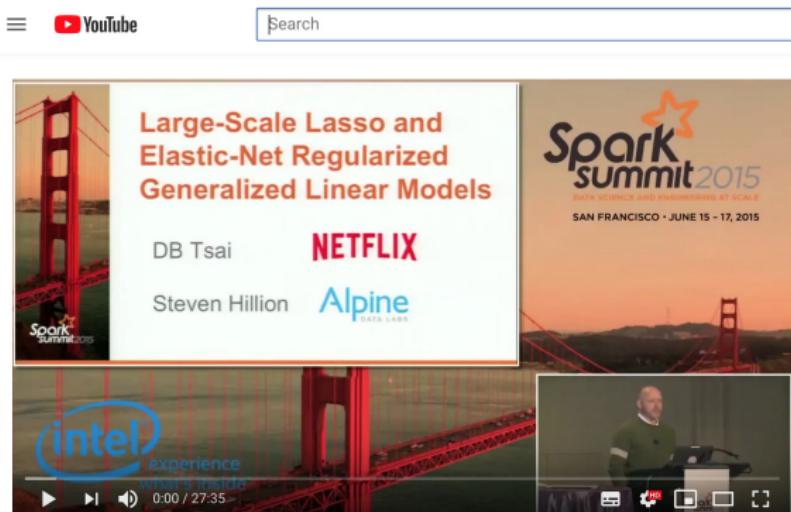
The screenshot shows the Apache Spark API Reference page. At the top, there's a navigation bar with links to 'Getting Started', 'User Guide', 'API Reference' (which is the active page), 'Development', and 'Migration Guide'. Below the navigation bar is a search bar labeled 'Search the docs ...'. To the left, there's a sidebar with a 'Spark' logo and a list of categories: 'Spark SQL', 'Pandas API on Spark', 'Structured Streaming', 'MLlib (DataFrame-based)', 'Spark Streaming', 'MLlib (RDD-based)', 'Spark Core', and 'Resource Management'. The main content area is titled 'API Reference' and contains a sub-header: 'This page lists an overview of all public PySpark modules, classes, functions and methods.' Below this, there's a hierarchical list of API sections:

- **Spark SQL**
 - Core Classes
 - Spark Session APIs
 - Configuration
 - Input and Output
 - DataFrame APIs
 - Column APIs
 - Data Types
 - Row
 - Functions
 - Window
 - Grouping
 - Catalog APIs
- **Pandas API on Spark**
 - Input/Output
 - General functions
 - Series
 - DataFrame
 - Index objects
 - Window
 - GroupBy
 - Machine Learning utilities
 - Extensions
- **Structured Streaming**
 - Core Classes
 - Input and Output
 - Query Management
- **MLlib (DataFrame-based)**
 - Pipeline APIs
 - Parameters
 - Feature

<https://spark.apache.org/docs/latest/api/python/reference/index.html>

References used in this lecture

Youtube video: *Large-Scale Lasso and Elastic-Net Regularized Generalized Linear Models* by DB Tsai (from Netflix) and Steven Hillion (from Alpine Data Labs).



2015-06-15 Large-Scale Elastic-Net Regularized Generalized Linear Models at Spark Summit
2015

<https://www.youtube.com/watch?v=TJer4ibUZ0I>

References used in this lecture

Paper: *Map-Reduce for Machine Learning on Multicore* by C-T Chu et al.
(2007).

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu *

chengtao@stanford.edu

Sang Kyun Kim *

skkim38@stanford.edu

Yi-An Lin *

ianl@stanford.edu

YuanYuan Yu *

yuanyuan@stanford.edu

Gary Bradski [#]

garybradski@gmail

Andrew Y. Ng *

ang@cs.stanford.edu

Kunle Olukotun *

kunle@cs.stanford.edu

* CS. Department, Stanford University 353 Serra Mall,

Stanford University, Stanford CA 94305-9025.

[#] Rexec Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain “summation form,” which allows them to be easily parallelized on multicore computers. We adapt Google’s map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.

References used in this lecture

Paper: *Optimization Methods for Large Scale Machine Learning* by L. Bottou et al. (2018).

Optimization Methods for Large-Scale Machine Learning

Leon Bottou*

Frank E. Curtis†

Jorge Nocedal‡

February 12, 2018

Abstract

This paper provides a review and commentary on the past, present, and future of numerical optimization algorithms in the context of machine learning applications. Through case studies on text classification and the training of deep neural networks, we discuss how optimization problems arise in machine learning and what makes them challenging. A major theme of our study is that large-scale machine learning represents a distinctive setting in which the stochastic gradient (SG) method has traditionally played a central role while conventional gradient-based nonlinear optimization techniques typically falter. Based on this viewpoint, we present a comprehensive theory of a straightforward, yet versatile SG algorithm, discuss its practical behavior, and highlight opportunities for designing algorithms with improved performance. This leads to a discussion about the next generation of optimization methods for large-scale machine learning, including an investigation of two main streams of research on techniques that diminish noise in the stochastic directions and methods that make use of second-order derivative approximations.

Scalable generalized linear models

Mauricio A. Álvarez, PhD

Scalable Machine Learning,
University of Sheffield



The
University
Of
Sheffield.

Contents

The exponential family

Generalized linear models

Iteratively reweighted least squares (IRLS)

Generalized linear models in Spark ML

References

Contents

The exponential family

Generalized linear models

Iteratively reweighted least squares (IRLS)

Generalized linear models in Spark ML

References

Introduction

- You are probably familiar by now with different types of probability distributions: the Gaussian, the Bernoulli, the Poisson, the Gamma, etc.
- It turns out that most of these are members of a broader class of distributions known as the exponential family.

Why the exponential family is important?

- It can be shown that the exponential family is the only family of distributions with finite-sized sufficient statistics.
- The exponential family is the only family of distributions for which conjugate priors exist.
- The exponential family can be shown to be the family of distributions that makes the least set of assumptions subject to some user-chosen constraints.
- The exponential family is at the core of generalized linear models.

Definition

- It is said that a pdf or a pmf $p(\mathbf{x}|\theta)$, with $\mathbf{x} \in \mathbb{R}^p$ and $\theta \in \mathbb{R}^d$, is in the **exponential family** if it is of the form

$$p(\mathbf{x}|\theta) = \frac{1}{Z(\theta)} h(\mathbf{x}) \exp [\theta^\top \phi(\mathbf{x})],$$

where

$$Z(\theta) = \int h(\mathbf{x}) \exp [\theta^\top \phi(\mathbf{x})] d\mathbf{x}.$$

- θ are known as the **natural parameters** or **canonical parameters**.
- $\phi(\mathbf{x}) \in \mathbb{R}^d$ is called a vector of **sufficient statistics**.
- $Z(\theta)$ is known as the **partition function**.
- $h(\mathbf{x})$ is a scaling constant, often 1.

Definition

- Distributions in the exponential family can also be expressed as

$$p(\mathbf{x}|\boldsymbol{\theta}) = h(\mathbf{x}) \exp [\boldsymbol{\theta}^\top \phi(\mathbf{x}) - A(\boldsymbol{\theta})],$$

where

$$A(\boldsymbol{\theta}) = \log Z(\boldsymbol{\theta}).$$

- $A(\boldsymbol{\theta})$ is called the **log partition function** or **cumulant function**.
- If $\phi(\mathbf{x}) = \mathbf{x}$, we say it is a **natural exponential family**.

Example: Bernoulli (I)

- For the Bernoulli distribution, $x \in \{0, 1\}$, and we have

$$p(x|\mu) = \text{Bern}(x|\mu) = \mu^x(1-\mu)^{1-x},$$

where $\mu = p(x=1)$.

- The expected value of x is equal to $\mathbb{E}[x] = \mu$.
- The distribution above can be written as

$$\begin{aligned} p(x|\mu) &= \exp\{x \log \mu + (1-x) \log(1-\mu)\}, \\ &= (1-\mu) \exp\left\{\log\left(\frac{\mu}{1-\mu}\right)x\right\} \end{aligned}$$

Example: Bernoulli (II)

- Comparing terms with the general expression for the exponential family, we observe that

$$\theta = \log\left(\frac{\mu}{1-\mu}\right), \quad \mu = \sigma(\theta) = \frac{1}{1 + \exp(-\theta)}.$$

- This means

$$Z(\theta) = \frac{1}{1-\mu} = \frac{1}{1 - \frac{1}{1+\exp(-\theta)}} = \frac{1}{\frac{\exp(-\theta)}{1+\exp(-\theta)}} = \frac{1}{\frac{1}{\exp(\theta)+1}} = \frac{1}{\sigma(-\theta)}.$$

- The Bernoulli distribution can be written as $p(x|\theta) = \sigma(-\theta) \exp(\theta x)$.
- The term $\theta = \log\left(\frac{\mu}{1-\mu}\right)$ is known as the **log-odds ratio**.

Example: univariate Gaussiana distribution

- The univariate Gaussian can be written in exponential family form as

$$\begin{aligned}\mathcal{N}(x|\mu, \sigma^2) &= \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left[-\frac{1}{2\sigma^2}(x-\mu)^2\right] \\ &= \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left[-\frac{1}{2\sigma^2}x^2 + \frac{\mu}{\sigma^2}x - \frac{1}{2\sigma^2}\mu^2\right] \\ &= \frac{1}{Z(\theta)} \exp(\theta^\top \phi(x)),\end{aligned}$$

where

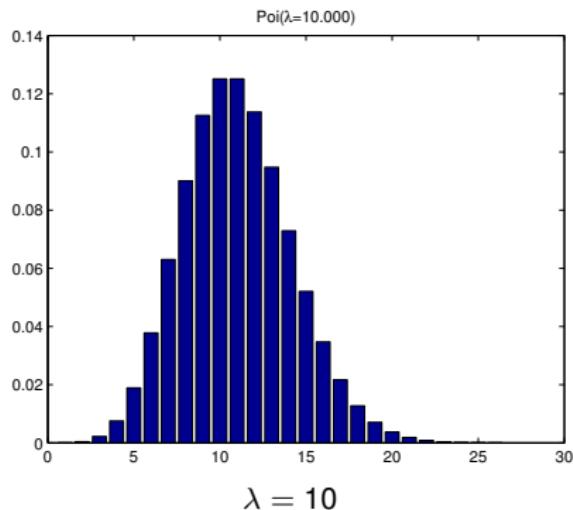
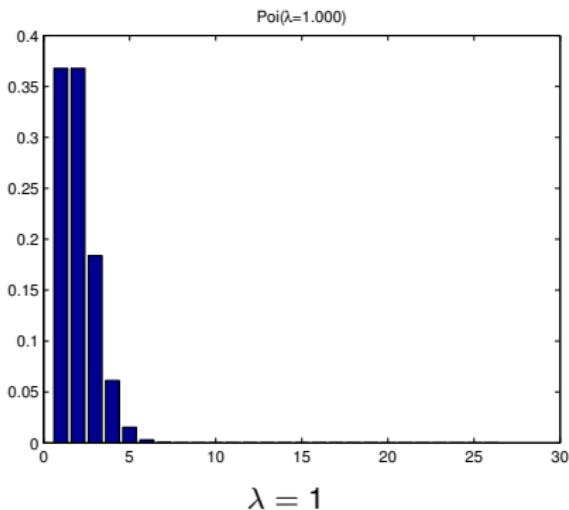
$$\begin{aligned}\theta &= \begin{bmatrix} \mu/\sigma^2 \\ -\frac{1}{2\sigma^2} \end{bmatrix}, & \phi(x) &= \begin{bmatrix} x \\ x^2 \end{bmatrix}, \\ Z(\theta) &= \sqrt{2\pi}\sigma \exp\left\{\frac{\mu^2}{2\sigma^2}\right\}, \\ A(\theta) &= -\frac{\theta_1^2}{4\theta_2} - \frac{1}{2}\log(-2\theta_2) - \frac{1}{2}\log(2\pi).\end{aligned}$$

Example: Poisson distribution (I)

- The Poisson distribution follows as

$$\text{Poi}(x|\lambda) = \exp(-\lambda) \frac{\lambda^x}{x!},$$

where $\lambda > 0$, and $x \in \{0, 1, 2, \dots\}$.



Example: Poisson distribution (II)

- As a member of the exponential family, it can be written as

$$\text{Poi}(x|\lambda) = \frac{h(x)}{Z(\theta)} \exp(\theta x),$$

where $\theta = \log \lambda$, $h(x) = 1/x!$, and $Z(\theta) = \exp(\lambda)$. Also, $A(\theta) = \lambda$.

- Recall that the expected value of x is equal to $\mathbb{E}[x] = \lambda$.
- Then, the mean parameter λ can be recovered from the canonical parameter using

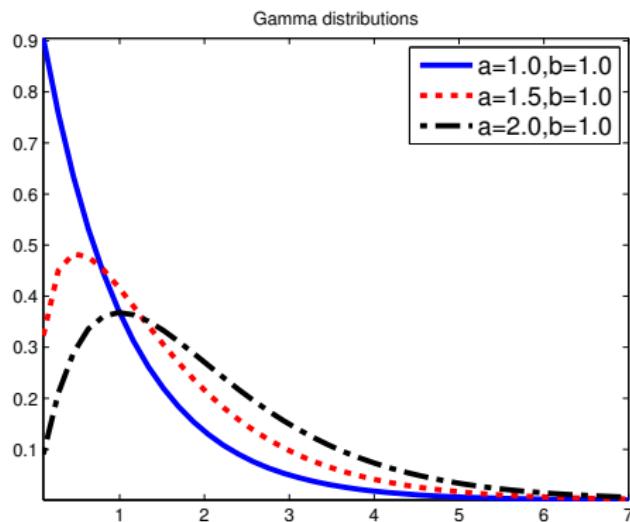
$$\lambda = \exp(\theta).$$

Example: Gamma distribution (I)

- The Gamma distribution follows as

$$\text{Ga}(x|a,b) = \frac{b^a}{\Gamma(a)} x^{a-1} \exp(-bx),$$

where $a > 0$ (shape), and $b > 0$ (rate). $\Gamma(a) = \int_0^\infty u^{a-1} e^{-u} du$ is the Gamma function.



Example: Gamma distribution (II)

- As a member of the exponential family, it can be written as

$$\text{Ga}(x|a, b) = \frac{1}{Z(\theta)} \exp(\theta^\top \phi(x)),$$

where

$$\theta = \begin{bmatrix} a - 1 \\ -b \end{bmatrix}, \quad \phi(x) = \begin{bmatrix} \log x \\ x \end{bmatrix},$$
$$Z(\theta) = \frac{\Gamma(a)}{b^a}, \quad A(\theta) = \log \Gamma(a) - a \log b.$$

Contents

The exponential family

Generalized linear models

Iteratively reweighted least squares (IRLS)

Generalized linear models in Spark ML

References

Definition

- Linear and logistic regression are examples of generalized linear models, or GLM.
- These are models in which the output density is in the exponential family.
- The mean parameters are a linear combination of the inputs, passed through a possibly nonlinear function, such as the logistic function.

General form (I)

- We previously used x as the variable we were modelling.
- Here we use y since the variable we want to model is the response variable.
- We want to model the relationship between a response variable y_i , and an input vector \mathbf{x}_i .
- Let us first consider the case of an unconditional distribution for the response variable

$$p(y_i|\theta, \sigma^2) = \exp \left[\frac{y_i\theta - A(\theta)}{\sigma^2} + c(y_i, \sigma^2) \right],$$

where σ^2 is the **dispersion parameter**, θ is the natural parameter, A is the partition function, and c is the normalisation constant. Usually, $\sigma^2 = 1$.

- The expression for $p(y_i|\theta, \sigma^2)$ looks similar to the exponential family.

General form (II)

- For example, in logistic regression, θ is the log-odds ratio

$$\theta = \log \left(\frac{\mu}{1 - \mu} \right),$$

where $\mu = \mathbb{E}[y] = P(y = 1)$ is the mean parameter.

- To convert from the mean parameter to the natural parameter, we can use a function ψ , $\theta = \psi(\mu)$.
- ψ is uniquely determined by the form of the exponential family distribution.
- The mapping is invertible, so that $\mu = \psi^{-1}(\theta)$.

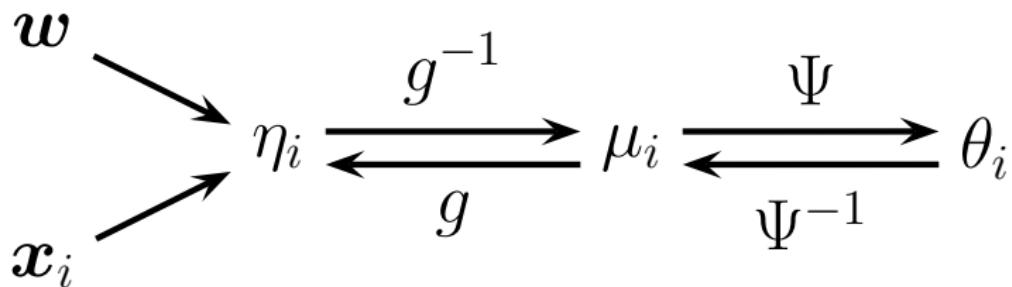
General form (III)

- Now let us add inputs or covariates.
- We first define a linear function of the inputs $\eta_i = \mathbf{w}^\top \mathbf{x}_i$.
- We now make the mean of the distribution be some invertible monotonic function of this linear combination.
- By convention, this function, known as the **mean function**, is denoted by g^{-1} , so

$$\mu_i = g^{-1}(\eta_i) = g^{-1}(\mathbf{w}^\top \mathbf{x}_i).$$

- The inverse of the mean function, namely $g()$, is called the **link function**.

Relationships between functions



$g^{-1}()$ is the **mean function**. $g()$ is the **link function**.

Link function

- We are free to choose almost any function we like for g , so long as it is invertible, and so long as g^{-1} has the appropriate range.
- For example, in logistic regression, we set $\mu_i = g^{-1}(\eta_i) = \sigma(\eta_i)$.

GLM with canonical link function

- One particularly simple form of link function is to use $g = \psi$.
- This is called the **canonical link function**.
- In this case $\theta_i = \eta_i = \mathbf{w}^\top \mathbf{x}_i$, so the model becomes

$$p(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2) = \exp \left[\frac{y_i \mathbf{w}^\top \mathbf{x}_i - A(\mathbf{w}^\top \mathbf{x}_i)}{\sigma^2} + c(y_i, \sigma^2) \right].$$

Canonical link functions $g = \psi$ for some GLMs

Distribution	Link $g(\mu)$	$\theta = \psi(\mu)$	$\mu = \psi^{-1}(\theta)$
$\mathcal{N}(\mu, \sigma^2)$	identity	$\theta = \mu$	$\mu = \theta$
Ber(μ)	logit	$\theta = \log\left(\frac{\mu}{1-\mu}\right)$	$\mu = \sigma(\theta)$
Poi(μ)	log	$\theta = \log(\mu)$	$\mu = \exp(\theta)$
Ga(a, b)	inverse	$\theta = \mu^{-1}$	$\mu = \theta^{-1}$.

Mean and variance of the response variable

- It can be shown that

$$\mathbb{E}[y|\mathbf{x}_i, \mathbf{w}, \sigma^2] = \mu_i = A'(\theta_i)$$

$$\text{var}[y|\mathbf{x}_i, \mathbf{w}, \sigma^2] = \sigma_i^2 = A''(\theta_i)\sigma^2.$$

Example: linear regression

- In linear regression, the response variable follows a normal distribution,

$$\begin{aligned} p(y_i|\mu_i, \sigma^2) &= \mathcal{N}(y_i|\mu_i, \sigma^2) \\ &= \frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}} \exp\left[-\frac{1}{2\sigma^2}(y_i - \mu_i)^2\right] \\ &= \exp\left[\frac{y_i\mu_i - \frac{\mu_i^2}{2}}{\sigma^2} - \frac{y_i^2}{2\sigma^2} + \log\left(\frac{1}{(2\pi\sigma^2)^{\frac{1}{2}}}\right)\right] \\ &= \exp\left[\frac{y_i\mu_i - \frac{\mu_i^2}{2}}{\sigma^2} - \frac{1}{2}\left(\frac{y_i^2}{\sigma^2} + \log(2\pi\sigma^2)\right)\right]. \end{aligned}$$

- For linear regression, $y_i \in \mathbb{R}$.
- The link function is the identity $\theta_i = \mu_i = \mathbf{w}^\top \mathbf{x}_i$.
- With $A(\mu_i) = \mu_i^2/2$, $\mathbb{E}[y_i] = \mu_i$, and $\text{var}[y_i] = \sigma^2$.

Example: logistic regression

- In logistic regression, the response variable follows a Bernoulli distribution

$$\begin{aligned} p(y_i | \mu_i, \sigma^2) &= \mu_i^{y_i} (1 - \mu_i)^{1-y_i} \\ &= \exp \left[\log \left(\frac{\mu_i}{1 - \mu_i} \right) y_i - (-\log(1 - \mu_i)) \right]. \end{aligned}$$

- The link function is the logit, $\log \left(\frac{\mu_i}{1 - \mu_i} \right) = \mathbf{w}^\top \mathbf{x}_i$.
- With $A(\theta_i) = -\log(1 - \sigma(\theta_i))$, $\mathbb{E}[y_i] = \sigma(\theta_i)$, and $\text{var}[y_i] = \sigma(\theta_i)(1 - \sigma(\theta_i))$.

Example: Poisson regression

- In Poisson regression, the response variable follows a Poisson distribution

$$p(y_i | \mathbf{x}_i, \mathbf{w}, \sigma^2) = \exp[y_i \log(\mu_i) - \mu_i - \log(y_i!)].$$

- The link function is log, $\log \mu_i = \mathbf{w}^\top \mathbf{x}_i$.
- With $A(\theta_i) = \exp(\theta_i)$, $\mathbb{E}[y_i] = \exp(\theta_i) = \lambda_i$.

Contents

The exponential family

Generalized linear models

Iteratively reweighted least squares (IRLS)

Generalized linear models in Spark ML

References

Log-likelihood for a GLM

- One of the appealing properties of GLMs is that they can be fit using exactly the same methods that we used to fit logistic regression.
- In particular, the log-likelihood has the following form

$$\ell(\mathbf{w}) = \frac{1}{\sigma^2} \sum_{i=1}^N \ell_i = \frac{1}{\sigma^2} \sum_{i=1}^N [\theta_i y_i - A(\theta_i)],$$

where $\ell_i = \theta_i y_i - A(\theta_i)$.

Gradient for the log-likelihood

- We can compute the gradient vector using the chain rule as follows

$$\begin{aligned}\frac{\partial \ell_i}{\partial w_j} &= \frac{d\ell_i}{d\theta_i} \frac{d\theta_i}{d\mu_i} \frac{d\mu_i}{d\eta_i} \frac{\partial \eta_i}{\partial w_j} \\ &= (y_i - A'(\theta_i)) \frac{d\theta_i}{d\mu_i} \frac{d\mu_i}{d\eta_i} x_{i,j} \\ &= (y_i - \mu_i) \frac{d\theta_i}{d\mu_i} \frac{d\mu_i}{d\eta_i} x_{i,j}.\end{aligned}$$

- If we use a canonical link, $\theta_i = \eta_i$, this simplifies to

$$\mathbf{g}(\mathbf{w}) = \frac{1}{\sigma^2} \left[\sum_{i=1}^N (y_i - \mu_i) \mathbf{x}_i \right] = \frac{1}{\sigma^2} \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}),$$

where $\boldsymbol{\mu} = [\mu_1, \dots, \mu_N]^\top$.

- This can be used inside a (stochastic) gradient descent procedure.

Hessian for the log-likelihood

- For improved efficiency, we could use a second-order method.
- If we use a canonical link, the Hessian is given by

$$\mathbf{H}(\mathbf{w}) = -\frac{1}{\sigma^2} \sum_{i=1}^N \frac{d\mu_i}{d\theta_i} \mathbf{x}_i \mathbf{x}_i^\top = -\frac{1}{\sigma^2} \mathbf{X}^\top \boldsymbol{\Sigma} \mathbf{X},$$

where $\boldsymbol{\Sigma} = \text{diag} \left(\frac{d\mu_1}{d\theta_1}, \dots, \frac{d\mu_N}{d\theta_N} \right)$.

- This can be used inside the Iterative Reweighted Least Squares (IRLS) algorithm.

Iterative reweighted least squares (IRLS) algorithm

- ❑ The Iterative Reweighted Least Squares algorithm is a particular case of the Newton's method.
- ❑ The updated parameters are obtained by iteratively solving a weighted least squares problem.

A least squares problem

- Remember that a least squares (LS) problem refers to

$$LS(\mathbf{w}) = \min_{\mathbf{w}} \sum_{i=1}^N (y_i - \mathbf{w}^\top \mathbf{x}_i)^2,$$

for a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i\}_{i=1}^N = \{\mathbf{X}, \mathbf{y}\}$.

- It can be shown that the vector \mathbf{w} that minimises $LS(\mathbf{w})$ is given as

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{y}.$$

A weighted least squares problem

- A weighted least squares (*WLS*) problem refers to

$$WLS(\mathbf{w}) = \min_{\mathbf{w}} \sum_{i=1}^N r_i(y_i - \mathbf{w}^\top \mathbf{x}_i)^2,$$

for a dataset $\mathcal{D} = \{\mathbf{x}_i, y_i, r_i\}_{i=1}^N = \{\mathbf{X}, \mathbf{R}, \mathbf{y}\}$, with $\mathbf{R} = \text{diag}(r_1, \dots, r_N)$.

- It can be shown that the vector \mathbf{w} that minimises $WLS(\mathbf{w})$ is given as

$$\mathbf{w} = (\mathbf{X}^\top \mathbf{R} \mathbf{X})^{-1} \mathbf{X}^\top \mathbf{R} \mathbf{y}.$$

Iterative reweighted least squares problem

- Newton's method for the log-likelihood of the GLM follows as

$$\begin{aligned}\mathbf{w}_{k+1} &= \mathbf{w}_k - \mathbf{H}_k^{-1} \mathbf{g}_k \\&= \mathbf{w}_k + (\mathbf{X}^\top \Sigma_k \mathbf{X})^{-1} \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}_k) \\&= (\mathbf{X}^\top \Sigma_k \mathbf{X})^{-1} [\mathbf{X}^\top \Sigma_k \mathbf{X} \mathbf{w}_k + \mathbf{X}^\top (\mathbf{y} - \boldsymbol{\mu}_k)] \\&= (\mathbf{X}^\top \Sigma_k \mathbf{X})^{-1} \mathbf{X}^\top \Sigma_k \mathbf{z}_k,\end{aligned}$$

where $\mathbf{z}_k = \mathbf{X} \mathbf{w}_k + \Sigma_k^{-1} (\mathbf{y} - \boldsymbol{\mu}_k)$ is known as the **working response**.

- At iteration k , the solution for \mathbf{w}_{k+1} has a similar form to the solution for a weighted least squared problem replacing \mathbf{R} for Σ_k , and \mathbf{y} for \mathbf{z}_k .
- The name IRLS is due to at each iteration, we solve a weighted least squares problem, where the weight matrix Σ_k changes at each iteration.

Contents

The exponential family

Generalized linear models

Iteratively reweighted least squares (IRLS)

Generalized linear models in Spark ML

References

GeneralizedLinearRegression()

- ❑ It uses IRLS (Iterative Reweighted Least Squares) for optimisation.
- ❑ It only allows ℓ_2 regularisation.
- ❑ Spark currently only supports up to 4096 features through its GeneralizedLinearRegression interface.
- ❑ It will throw an exception if this constraint is exceeded.

GLM available in Spark

- It includes the following families

Family	Response type	Supported links
Gaussian	Continuous	Identity*, Log, Inverse
Binomial	Binary	Logit*, Probit, CLogLog
Poisson	Count	Log*, Identity, Sqrt
Gamma	Continuous	Inverse*, Identity, Log
Tweedie	Zero-inflated continuous	Power link function

where * stands for canonical link.

- For any random variable Z that obeys a Tweedie distribution, the variance $\text{var}(Z)$ relates to the mean $\mathbb{E}(Z)$ by the power law:
 $\text{var}(Z) = a\mathbb{E}(Z)^p$, where a and p are positive constants.
- The parameters are set using `family` and `link`.

Parameters to adjust

- **maxIter**: max number of iterations.
- **regParam**: regularization parameter (≥ 0).
- **family**: name of family which is a description of the label distribution to be used in the model.
- **link**: name of link function which provides the relationship between the linear predictor and the mean of the distribution function.

Particular cases: `LinearRegression()`

- ❑ If your ‘family’ is Gaussian and the link function is the ‘identity’, your model is just equivalent to linear regression.
- ❑ My recommendation is that you use `LinearRegression()` instead of `GeneralizedLinearRegression()`.
- ❑ `LinearRegression()` allows for ℓ_1 , ℓ_2 and elastic net regularization through L-BFGS or OWL-QN.

Particular cases: LogisticRegression()

- ❑ If your ‘family’ is Binomial and the link function is ‘logit’, your model is equivalent to logistic regression.
- ❑ My recommendation is that you use `LogisticRegression()` instead of `GeneralizedLinearRegression()`.
- ❑ `LogisticRegression()` allows for ℓ_1 , ℓ_2 and elastic net regularization through L-BFGS or OWL-QN.

Contents

The exponential family

Generalized linear models

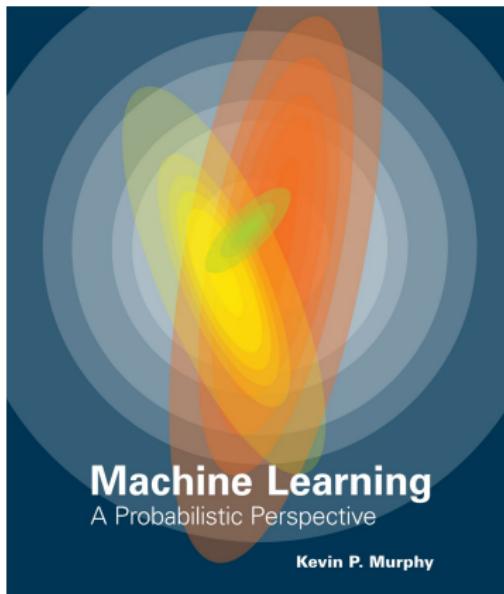
Iteratively reweighted least squares (IRLS)

Generalized linear models in Spark ML

References

References used in this lecture

Book: *Machine Learning: A Probabilistic Perspective* by Kevin P Murphy, 2012.



References used in this lecture

Website: *Spark Python API Documentation.*

The screenshot shows the official Apache Spark API documentation website. At the top, there's a dark header bar with the "Spark" logo on the left and navigation links: "Getting Started", "User Guide", "API Reference" (which is highlighted in red), "Development", and "Migration Guide". Below the header is a search bar with the placeholder "Search the docs ...". To the right of the search bar is a "API Reference" section title. Underneath it, a sub-section title "This page lists an overview of all public PySpark modules, classes, functions and methods." is followed by a large, nested list of API categories and sub-categories. The categories include "Spark SQL", "Pandas API on Spark", "Structured Streaming", "MLlib (DataFrame-based)", "MLlib (RDD-based)", "Spark Core", and "Resource Management". Each category has a corresponding list of sub-topics, such as "Core Classes", "Input/Output", "Series", etc.

- [Spark SQL](#)
 - [Core Classes](#)
 - [Spark Session APIs](#)
 - [Configuration](#)
 - [Input and Output](#)
 - [DataFrame APIs](#)
 - [Column APIs](#)
 - [Data Types](#)
 - [Row](#)
 - [Functions](#)
 - [Window](#)
 - [Grouping](#)
 - [Catalog APIs](#)
- [Pandas API on Spark](#)
 - [Input/Output](#)
 - [General functions](#)
 - [Series](#)
 - [DataFrame](#)
 - [Index objects](#)
 - [Window](#)
 - [GroupBy](#)
 - [Machine Learning utilities](#)
 - [Extensions](#)
- [Structured Streaming](#)
 - [Core Classes](#)
 - [Input and Output](#)
 - [Query Management](#)
- [MLlib \(DataFrame-based\)](#)
 - [Pipeline APIs](#)
 - [Parameters](#)
 - [Feature](#)

<https://spark.apache.org/docs/latest/api/python/reference/index.html>

References used in this lecture

Youtube video: *Generalized Linear Models in Spark MLlib and SparkR* by Xiangrui Meng (from Databricks).

The screenshot shows a YouTube video player interface. At the top, there is a navigation bar with a menu icon, the YouTube logo, and a search bar. The main video frame displays a presentation slide with a teal background and white text. The title of the slide is "Generalized Linear Models in Spark MLlib and SparkR". Below the title, it says "Xiangrui Meng joint with Joseph Bradley, Eric Liang, Yarbo Liang (MiningLamp), DB Tsai (Netflix), et.al." and "2016/02/17 - Spark Summit East". A "databricks" logo is visible in the bottom right corner of the slide. To the right of the slide, there is promotional text for "SPARK SUMMIT EAST DATA SCIENCE AND ENGINEERING AT SCALE FEBRUARY 16–18, 2016 NEW YORK CITY". Below the slide, there is a smaller video preview window showing a person speaking at a podium. The video player interface includes standard controls like play/pause, volume, and progress bar (0:21 / 29:05). At the bottom of the player, there is descriptive text: "Generalized Linear Models in Spark MLlib and SparkR", the number of views (2,521), and social sharing icons for likes (11), dislikes, share, save, and more.

https://www.youtube.com/watch?v=PSZW6hcQ_7w

References used in this lecture

Paper: *Map-Reduce for Machine Learning on Multicore* by C-T Chu et al.
(2007).

Map-Reduce for Machine Learning on Multicore

Cheng-Tao Chu *

chengtao@stanford.edu

Sang Kyun Kim *

skkim38@stanford.edu

Yi-An Lin *

ianl@stanford.edu

YuanYuan Yu *

yuanyuan@stanford.edu

Gary Bradski [#]

garybradski@gmail

Andrew Y. Ng *

ang@cs.stanford.edu

Kunle Olukotun *

kunle@cs.stanford.edu

* CS. Department, Stanford University 353 Serra Mall,

Stanford University, Stanford CA 94305-9025.

[#] Rexec Inc.

Abstract

We are at the beginning of the multicore era. Computers will have increasingly many cores (processors), but there is still no good programming framework for these architectures, and thus no simple and unified way for machine learning to take advantage of the potential speed up. In this paper, we develop a broadly applicable parallel programming method, one that is easily applied to *many* different learning algorithms. Our work is in distinct contrast to the tradition in machine learning of designing (often ingenious) ways to speed up a *single* algorithm at a time. Specifically, we show that algorithms that fit the Statistical Query model [15] can be written in a certain “summation form,” which allows them to be easily parallelized on multicore computers. We adapt Google’s map-reduce [7] paradigm to demonstrate this parallel speed up technique on a variety of learning algorithms including locally weighted linear regression (LWLR), k-means, logistic regression (LR), naive Bayes (NB), SVM, ICA, PCA, gaussian discriminant analysis (GDA), EM, and backpropagation (NN). Our experimental results show basically linear speedup with an increasing number of processors.

Scalable neural networks

Mauricio A. Álvarez, PhD

Scalable Machine Learning,
University of Sheffield



The
University
Of
Sheffield.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References

Introduction

- In MLAI you studied neural networks and automatic differentiation.
- In this lecture, we will talk about some topics at the interface of neural networks and Spark.
- This is a very active field at the moment, specially for combining deep learning with Spark.
- Several open source tools started developing in the last couple of years and their technical content gets updated very fast.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References

Feedforward neural networks

- In this section, we will briefly review **feedforward neural networks** or **multilayer perceptrons**.
- The MLP assumes that the input is a fixed-dimensional vector, say $\mathbf{x} \in \mathbb{R}^P$.
- This data is referred to as "**unstructured data**": there are not assumptions about the form of the input.
- Other types of neural networks assume that the input has variable size or shape
 - convolutional NN for variable-sized images.
 - recurrent NN for variable-sized sequences.
 - graph NN for variable-sized graphs.

The Iris flower data set

- ❑ The Iris flower dataset is a classic dataset used in pattern recognition.
- ❑ It was introduced by the British statistician Ronald Fisher in a paper from 1936.
- ❑ The dataset contains 150 instances
- ❑ Four input features: sepal length, sepal width, petal length, and petal width ($\mathbf{x} \in \mathbb{R}^4$).
- ❑ A categorical output, the species, which can take three values: Iris setosa, Iris virginica and Iris versicolor.

Iris flowers



Iris versicolor

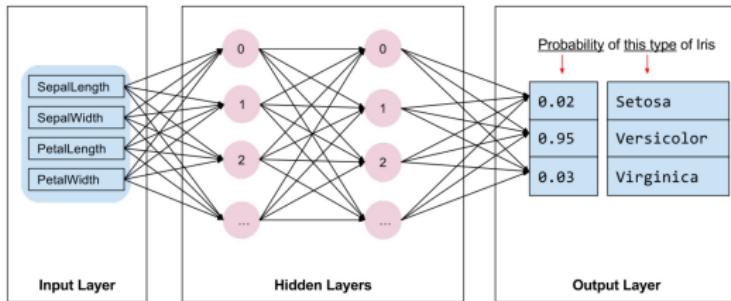


Iris virginica



Iris setosa

A 2-layer MLP applied to the Iris dataset



(a) A 2-layer MLP

$$p(y|\mathbf{x}; \theta) = \text{Cat}(y|f_3(\mathbf{x}; \theta))$$

$$f_3(\mathbf{x}; \theta) = \mathcal{S}(\mathbf{W}_3 f_2(\mathbf{x}; \theta) + \mathbf{b}_3)$$

$$f_2(\mathbf{x}; \theta) = \varphi_2(\mathbf{W}_2 f_1(\mathbf{x}; \theta) + \mathbf{b}_2)$$

$$f_1(\mathbf{x}; \theta) = \varphi_1(\mathbf{W}_1 f_0(\mathbf{x}; \theta) + \mathbf{b}_1)$$

$$f_0(\mathbf{x}; \theta) = \mathbf{x}$$

(b) Mathematical model

$\theta = (\mathbf{W}_3, \mathbf{b}_3, \mathbf{W}_2, \mathbf{b}_2, \mathbf{W}_1, \mathbf{b}_1)$, $\mathcal{S}(\cdot)$ soft-max function, $\text{Cat}(\cdot)$ categorical distribution, φ_1, φ_2 activation functions.

A L -layer MLP

- A L -layer MLP is in general defined as a non-linear function,

$$f(\mathbf{x}, \theta) = f_L(f_{L-1}(\cdots(f_1(\mathbf{x}))\cdots)),$$

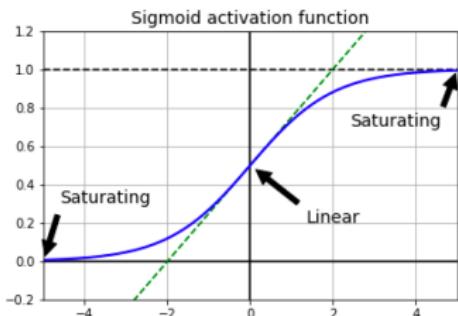
where $f_l(\mathbf{z}_{l-1}) = \varphi_l(\mathbf{W}_l \mathbf{z}_{l-1} + \mathbf{b}_l) = \mathbf{z}_l$, \mathbf{z}_{l-1} are the hidden units at layer $l - 1$ and $\varphi_l(\cdot)$ are the activation functions at layer l .

- In the expression above $\theta = [\mathbf{W}_L, \mathbf{b}_L, \cdots, \mathbf{W}_1, \mathbf{b}_1]$.
- The activation functions $\varphi_l(\cdot)$ are non-linear, differentiable functions such that $\varphi_l : \mathbb{R} \rightarrow \mathbb{R}$.

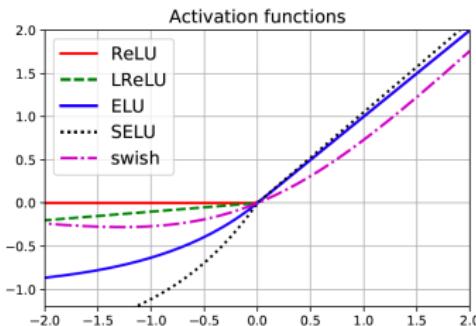
Activation functions φ_I

Name	Definition	Range
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$[0, 1]$
Hyperbolic tangent	$\tanh(a) = 2\sigma(2a) - 1$	$[-1, 1]$
Softplus	$\sigma_+(a) = \log(1 + e^a)$	$[0, \infty]$
Rectified linear unit	$\text{ReLU}(a) = \max(a, 0)$	$[0, \infty]$
Leaky ReLU	$\max(a, 0) + \alpha \min(a, 0)$	$[-\infty, \infty]$
Exponential linear unit	$\max(a, 0) + \min(\alpha(e^a - 1), 0)$	$[-\infty, \infty]$
Swish	$a\sigma(a)$	$[-\infty, \infty]$

(a) Mathematical form for different activation functions



(b) Sigmoid activation function



(c) Activation functions

Backpropagation

- ❑ Remember that the backpropagation algorithm is used to compute the gradient of the loss function wrt the parameters of the NN.
- ❑ It is an example of reverse-mode differentiation (Lecture 5 in MLAI).
- ❑ Once the gradient is computed, it is passed to a gradient-based optimization algorithm.

Objective functions

- Given a dataset $\mathcal{D} = \{\mathbf{x}_n, \mathbf{y}_n\}_{n=1}^N$, the NN parameters are usually fit using maximum likelihood estimation

$$\text{NLL}(\mathcal{D}; \theta) = -\log p(\mathcal{D}|\theta) = -\sum_{n=1}^N \log p(\mathbf{y}_n|\mathbf{x}_n; \theta).$$

- Practical considerations when training deep models:
 - turning the learning rate.
 - vanishing gradient problem. Solutions: modify the architecture (residual connections), standardize the activations at each layer (batch normalization) and careful parameter initialization.
 - exploding gradients. Addressed with gradient clipping.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References

MultilayerPerceptronClassifier (I)

- Spark ML implements feed-forward neural networks through the `MultilayerPerceptronClassifier` class.
- Nodes in the intermediate layers use the logistic sigmoid function

$$\varphi_l(z_i) = \frac{1}{1 + \exp(-z_i)}.$$

- Nodes in the output layer use the softmax function (a.k.a multinomial logit)

$$S(z_k) = \frac{\exp(z_k)}{\sum_{c=1}^K \exp(z_c)},$$

where K is the number of classes corresponding to the same number of nodes in the output layer.

MultilayerPerceptronClassifier (II)

- ❑ Spark ML uses backpropagation for learning the model.
- ❑ L-BFGS is the default optimization routine.
- ❑ The parameter `layers` specifies the number of layers and the number of nodes per layer, from input layer to output layer.
- ❑ E.g. if `layers = [780, 100, 10]`, this means a NN with 780 inputs, one hidden layer with 100 nodes and an output layer with 10 classes.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References

Beyond Spark ML

- ❑ Spark ML is not always the best solution for our machine learning needs.
 - Predictions that require super low-latency (e.g. real-time, less than 10ms)
 - It does not have support for the algorithm we would like to use, e.g. deep learning.
- ❑ In these cases, we can still use Spark, but not Spark ML.
- ❑ There are different alternatives to use external models inside Spark ML.
- ❑ For example, we can use Pandas UDFs (user-defined functions) for distributed inference of single-node models.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References

Deep learning and Spark

Three major ways to use deep learning in Spark

- ❑ Inference.
- ❑ Featurization and transfer learning.
- ❑ Model training.

Inference

- The simplest way to use deep learning is to use a pretrained model to make predictions in large datasets in parallel using Spark.
- For example, the InceptionV3 model, a large neural network model for object recognition can be applied to our own collection of images.
- In principle, one can simply use PySpark with a call to a framework like TensorFlow or Pytorch in a map function to get distributed inference.
- In practice, there are further optimizations that can be made to make such a call efficient.

Featurization and transfer learning

- We can also use an existing model as a *feature extractor* instead of taking its final output.
- As you probably know by now, deep learning models tend to learn useful feature representations in their lower layers.
- Low-level features learned in Imagenet can be used a starting point to learn new models in a different dataset.
- This method is called *transfer learning* and it is particularly useful when we do not have a large amount of training data for the new problem.

Model training

- ❑ Spark can also be used to train a new deep learning model from scratch.
- ❑ One can either train a *single* model over multiple executors communicating between them.
- ❑ Or one can train *multiple* instances of similar models in parallel to try various model architectures and hyperparameters.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References

What are UDFs?

- Spark SQL has a lot of built-in functions that can be applied to dataframes.
- Spark is flexible enough to allow users to define their own functions.
- These are known as *user-defined-functions* (UDFs).
- A typical example in data analytics:
 - wrap a ML model within an UDF
 - query its predictions in Spark SQL
 - there is no need to understand the internal behavior of the model.

PySpark UDFs before Spark 2.3

- ❑ Up until Spark 2.3, a permanent issue with PySpark UDFs was that they had lower performance compared with Scala UDFs.
- ❑ The reason for this was that PySpark UDFs required data movement between JVM and Python, which was expensive.
- ❑ For details, see [Speeding up PySpark with Apache Arrow](#) by Bryan Cutler.

Pandas UDFs from Spark 2.3

- ❑ Pandas UDFs, introduced in Spark 2.3, addressed this issue by using Apache Arrow to transfer data and Pandas to work with the data.
- ❑ Once the data is in Apache Arrow, there is no longer the need to serialize the data as it is already in a format consumable by the Python process.
- ❑ Instead of operating on individual inputs row by row, now the operation is over Pandas Series or DataFrame.
- ❑ This is what is known as vectorized execution.

A simple example of speed-up

```
from pyspark.sql.functions import rand
df = spark.range(1 << 22).toDF("id").withColumn("x", rand())
df.printSchema()
```

```
root
|-- id: long (nullable = false)
|-- x: double (nullable = false)
```

```
%time pdf = df.toPandas()
```

```
CPU times: user 10.6 s, sys: 714 ms, total: 11.3 s
Wall time: 16.1 s
```

```
spark.conf.set("spark.sql.execution.arrow.enabled", "true")
```

```
%time pdf = df.toPandas()
```

```
CPU times: user 102 ms, sys: 134 ms, total: 236 ms
Wall time: 1.04 s
```

```
pdf.describe()
```

	id	x
count	4.194304e+06	4.194304e+06

Pandas UDFs from Spark 3.0

- ❑ From Apache Spark 3.0 with Python 3.6+, the Pandas UDFs were split into two API categories: **Pandas UDFs** and Pandas Function APIs.
- ❑ Pandas UDFs
 - From Spark 3.0, Pandas UDFs infer the Pandas UDFs type from *Python type hints* such as `pandas.Series` or `pandas.Dataframes`.
 - Previously, users were required to manually define and specify each Pandas UDF type.
 - Currently, the supported cases of Python type hints in Pandas UDFs are
 - ▶ Series to Series
 - ▶ Iterator of Series to Iterator of Series
 - ▶ Iterator of multiple series to Interator of Series
 - ▶ Series to Scalar.
 - ▶ All of the above replacing Series with DataFrame.

Pandas UDFs from Spark 3.0

- ❑ From Apache Spark 3.0 with Python 3.6+, the Pandas UDFs were split into two API categories: Pandas UDFs and **Pandas Function APIs**.
- ❑ Pandas Function APIs
 - They allow to directly apply a local Python function to a PySpark Dataframe where both the input and output are Pandas instances.
 - Currently, the supported Pandas Function APIs are
 - ▶ Grouped Map with `groupBy().applyInPandas()`
 - ▶ Map with `mapInPandas()`
 - ▶ Co-grouped Map with `groupBy().cogroup().applyInPandas()`
- ❑ We will see an example of using `mapInPandas()` for making inference with a neural network model in this week's lab.
- ❑ [This blog post by Databricks](#) provides more details on Pandas UDFs and Pandas Function APIs in Apache Spark 3.0

Iterator support in Pandas UDFs

- ❑ A typical use case for Pandas UDFs is to load a machine learning or deep learning model to perform distributed inference.
- ❑ If the model is large (typical deep learning models can be tens or hundreds of Mbs), then there is a high overhead to load the same model again and again for every batch in the same Python process.
- ❑ Pandas UDFs accept iterators of `pandas.Series` or `pandas.DataFrame`.
- ❑ The pseudocode looks something like

```
@pandas_udf(...)  
def predict(iterator):  
    model = ... # load model  
    for features in iterator:  
        yield model.predict(features)
```

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

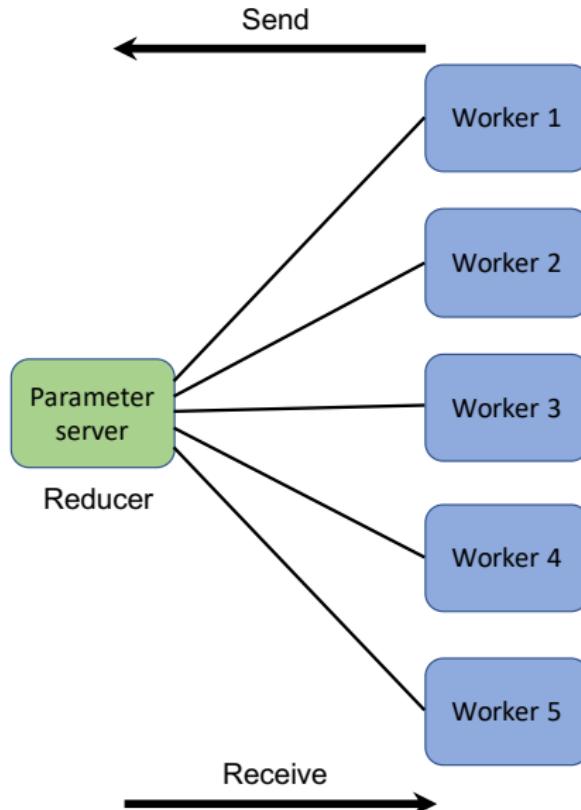
References

Horovod

- ❑ Horovod is an open-source software framework for distributed deep learning training using TensorFlow, Keras, PyTorch and MXNet.
- ❑ It was developed by Uber.
- ❑ Initial release was from August 2017 and the latest stable release is from May, 2020.
- ❑ Details on the following slides are from the blog post [Bringing HPC Techniques to Deep Learning](#) by Andrew Gibiansky.



The parameter server approach



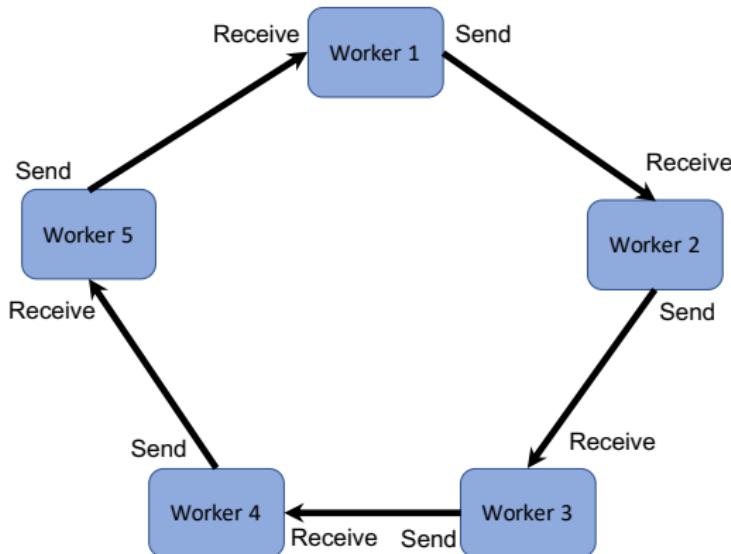
The parameter server approach: communication cost

- Deep learning models have potentially millions of parameters.
- For example, AlexNet has 60 millions of parameters. There are models today that have even billions of parameters.
- In the parameter server approach, each worker requires a complete copy of the neural network.
- Sending the parameters of these NNs back and forward between the server and the workers creates a communication overhead that slows training.
- The more workers in the system, the greater the communication cost.

Ring reduce

- ❑ In contrast, ring reduce is an algorithm for which communication is constant.
- ❑ If we only consider bandwidth as a factor in the communication cost, the ring allreduce is an optimal communication algorithm.
- ❑ The workers in ring reduce are arranged in a logical ring.

Ring allreduce

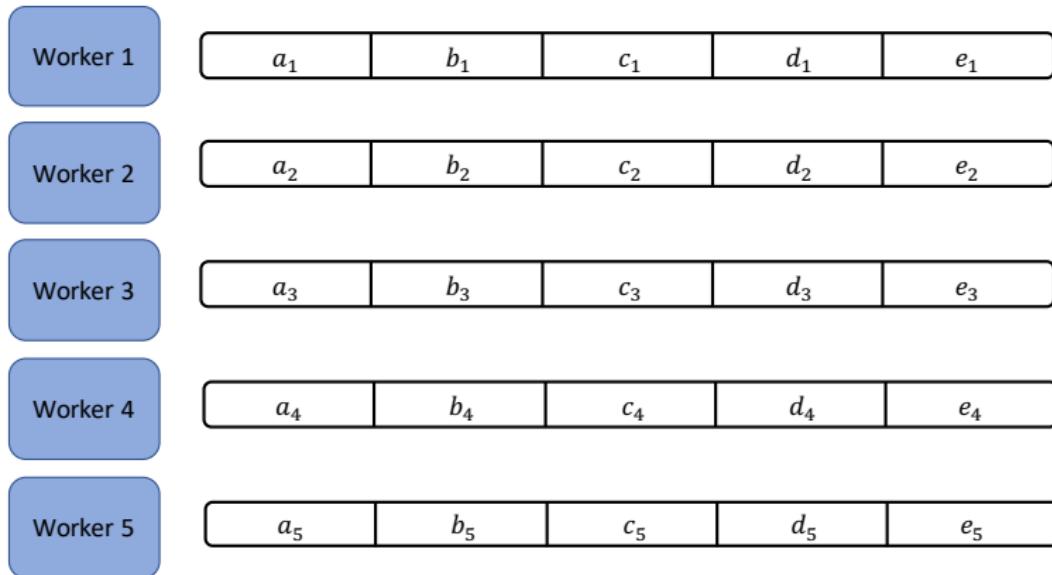


- Each worker has a right neighbour and a left neighbour.
- Each worker sends data to its right neighbour and receives data from its left neighbour.

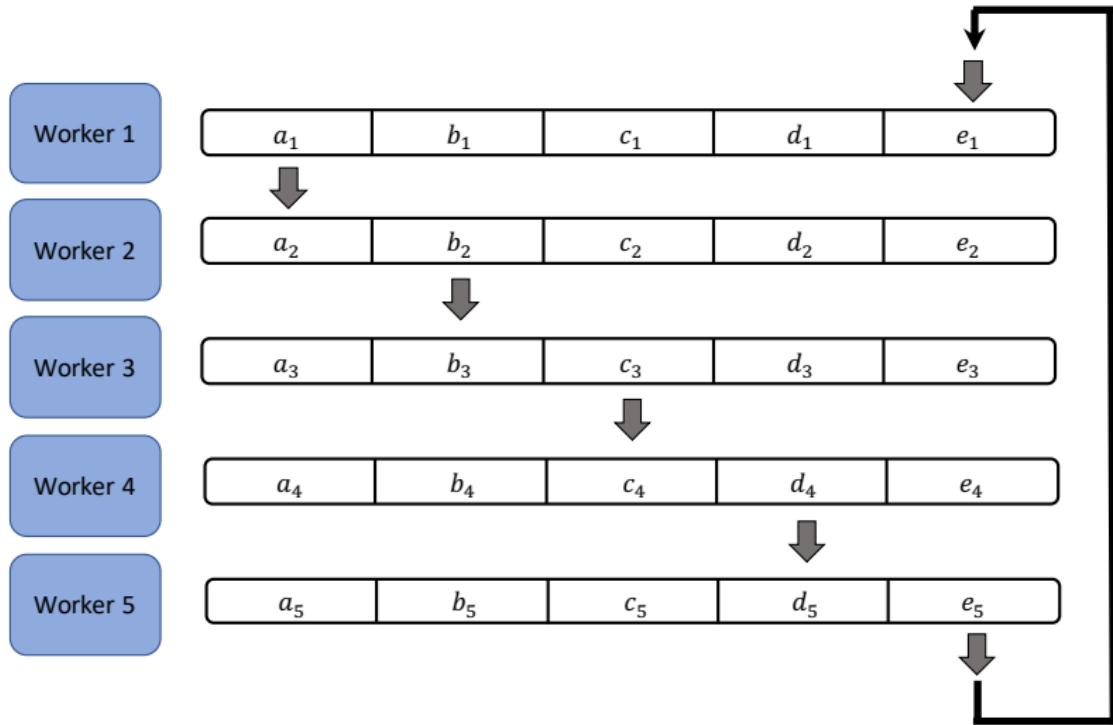
Ring allreduce: two-stages

- Ring all reduce is performed in two stages.
- In the scatter-reduce stage, workers exchange data such that every worker ends up with a chunk of the final result.
- In the allgather stage, the workers exchange those chunks such that each worker ends up with the complete final result.

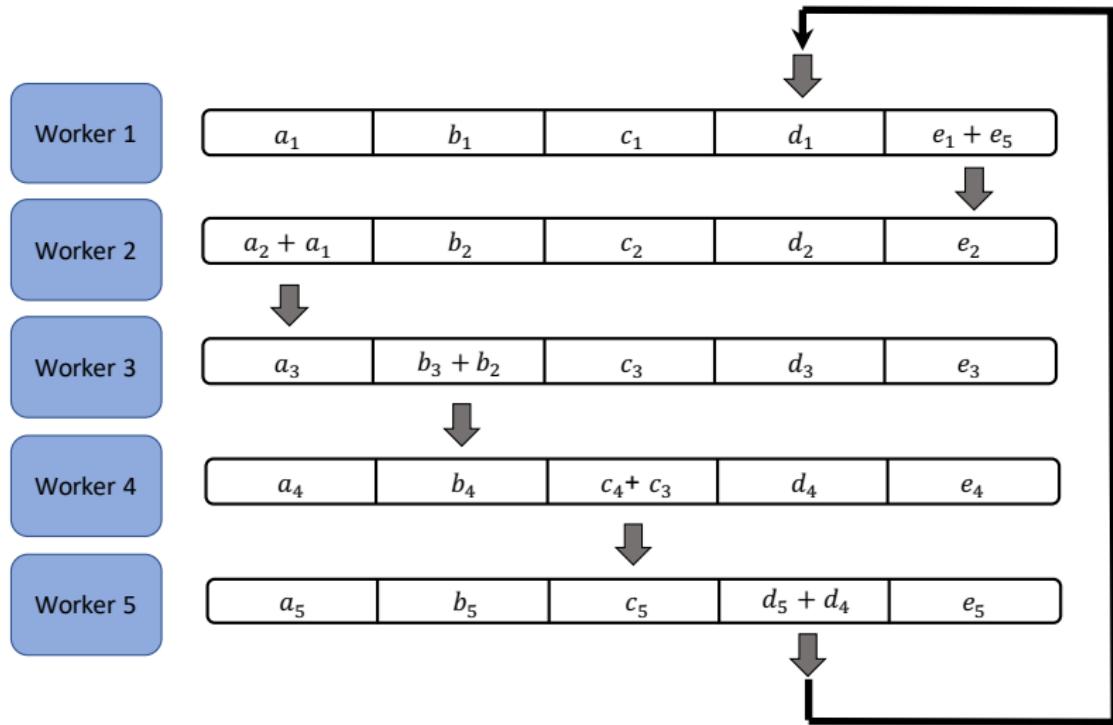
Summing arrays: scatter-reduce



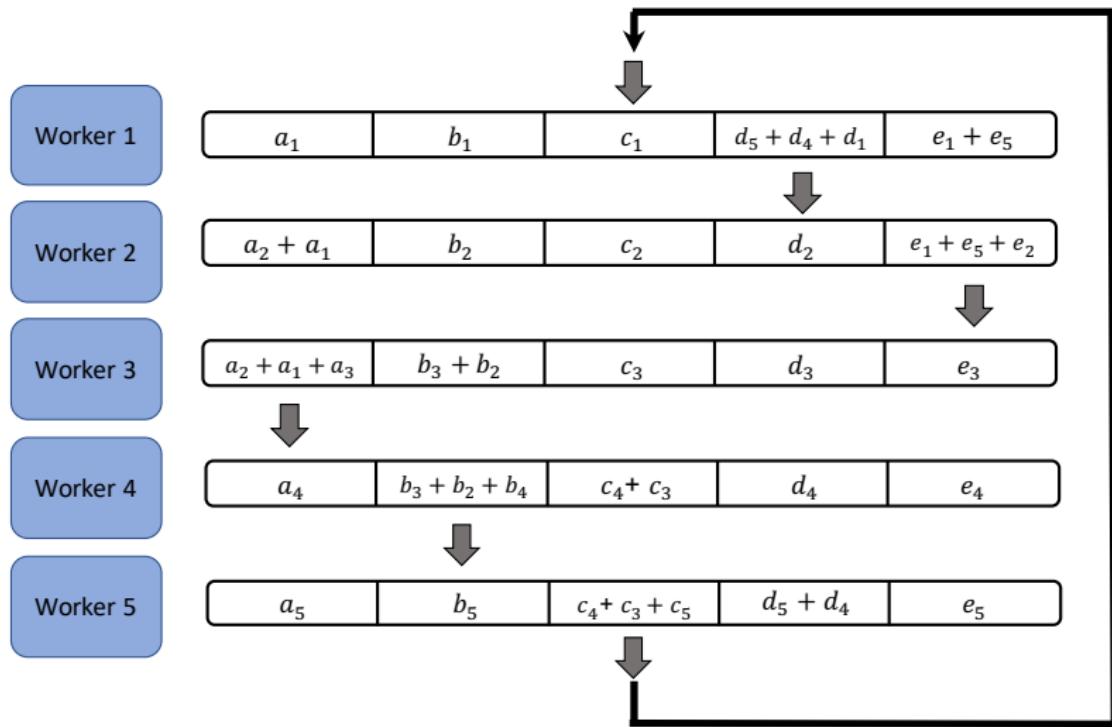
Summing arrays: scatter-reduce



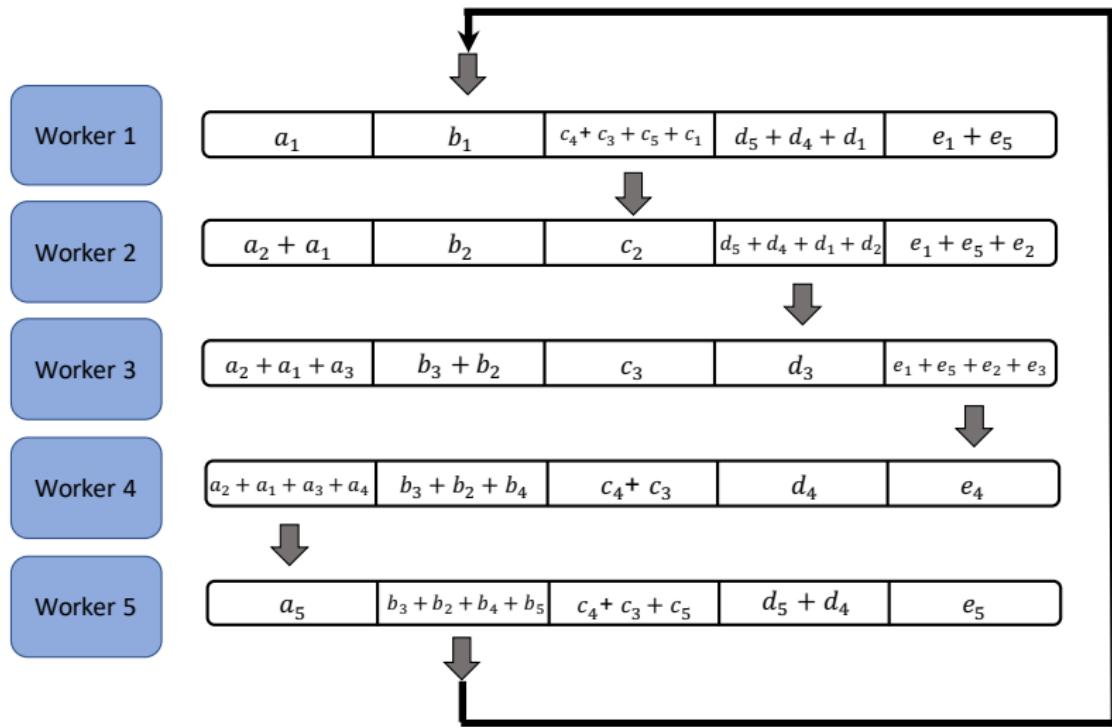
Summing arrays: scatter-reduce



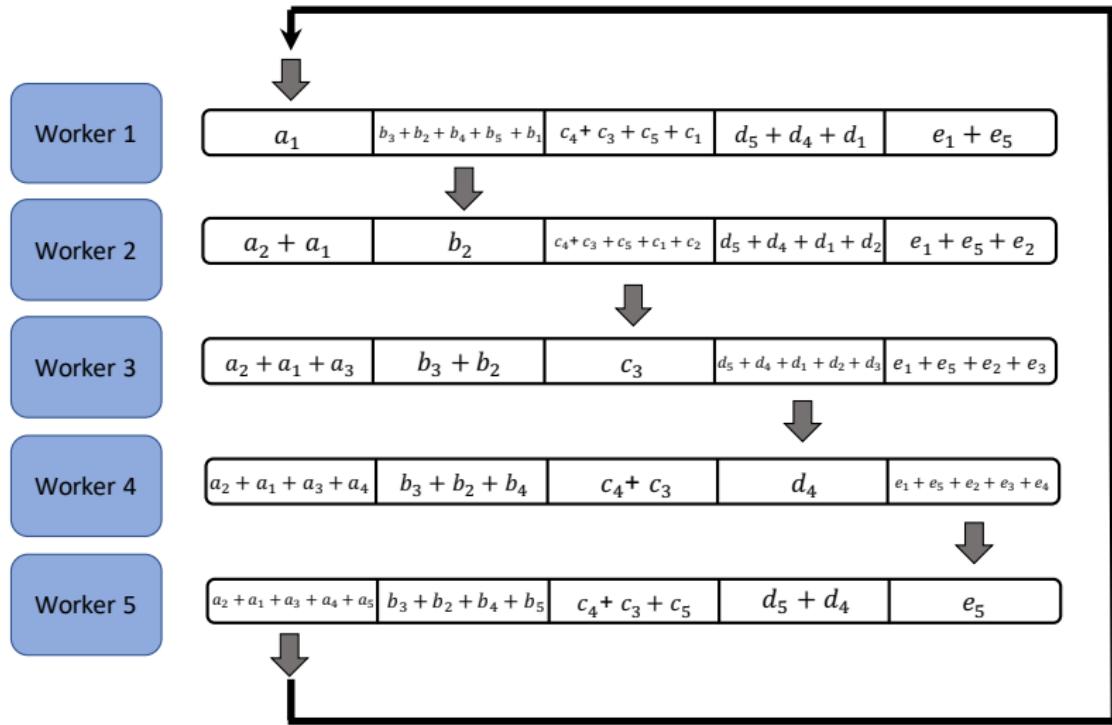
Summing arrays: scatter-reduce



Summing arrays: scatter-reduce



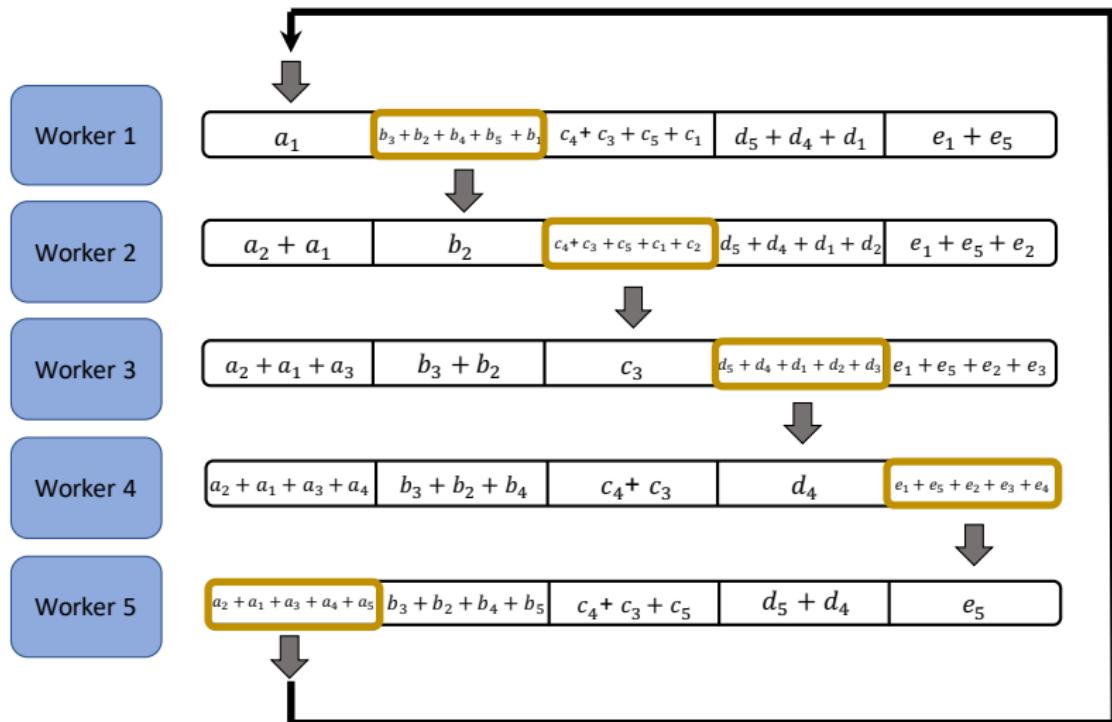
Summing arrays: scatter-reduce



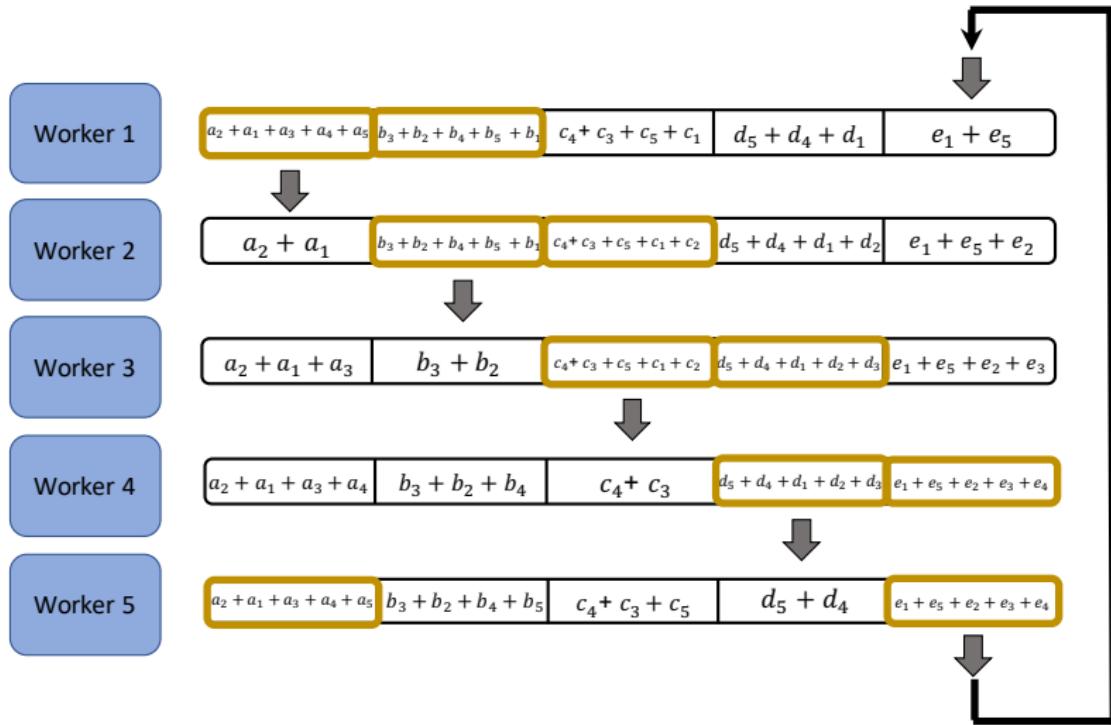
Summing arrays: scatter-reduce

Worker 1	a_1	$b_3 + b_2 + b_4 + b_5 + b_1$	$c_4 + c_3 + c_5 + c_1$	$d_5 + d_4 + d_1$	$e_1 + e_5$
Worker 2	$a_2 + a_1$	b_2	$c_4 + c_3 + c_5 + c_1 + c_2$	$d_5 + d_4 + d_1 + d_2$	$e_1 + e_5 + e_2$
Worker 3	$a_2 + a_1 + a_3$	$b_3 + b_2$	c_3	$d_5 + d_4 + d_1 + d_2 + d_3$	$e_1 + e_5 + e_2 + e_3$
Worker 4	$a_2 + a_1 + a_3 + a_4$	$b_3 + b_2 + b_4$	$c_4 + c_3$	d_4	$e_1 + e_5 + e_2 + e_3 + e_4$
Worker 5	$a_2 + a_1 + a_3 + a_4 + a_5$	$b_3 + b_2 + b_4 + b_5$	$c_4 + c_3 + c_5$	$d_5 + d_4$	e_5

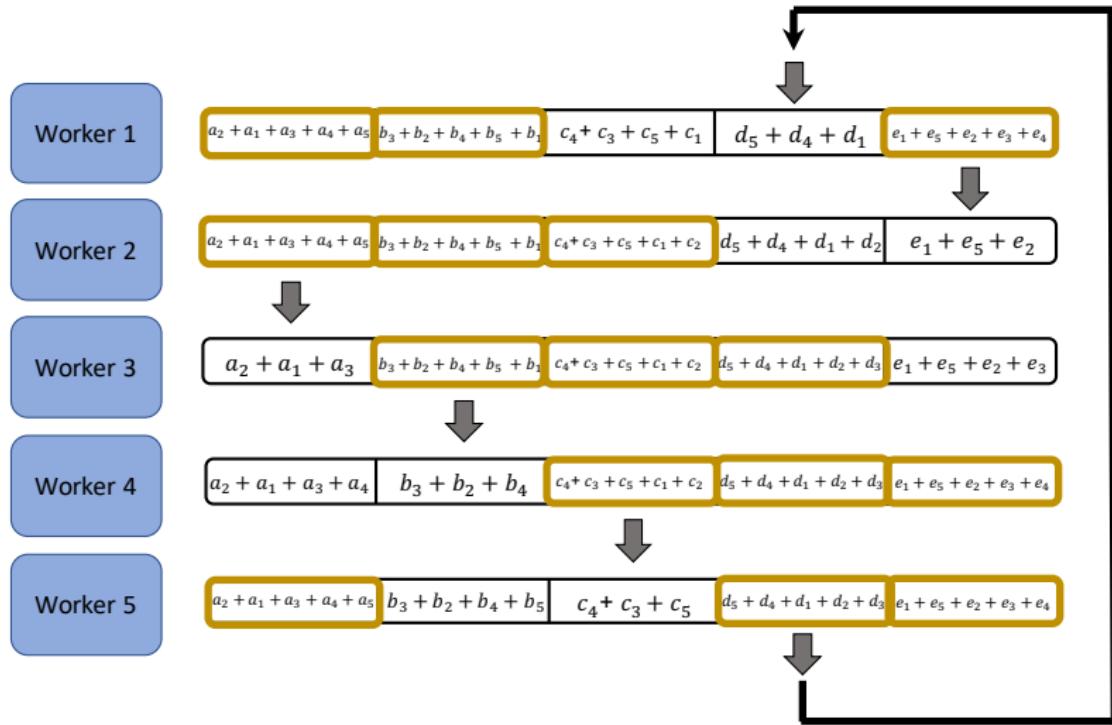
Summing arrays: allgather



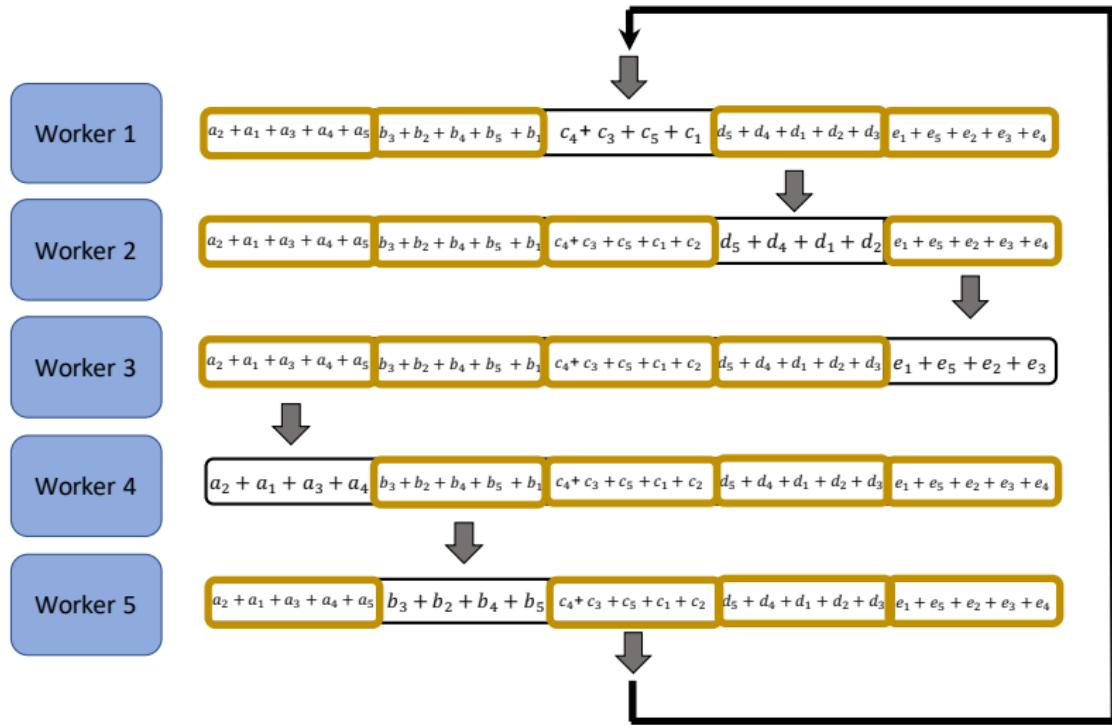
Summing arrays: allgather



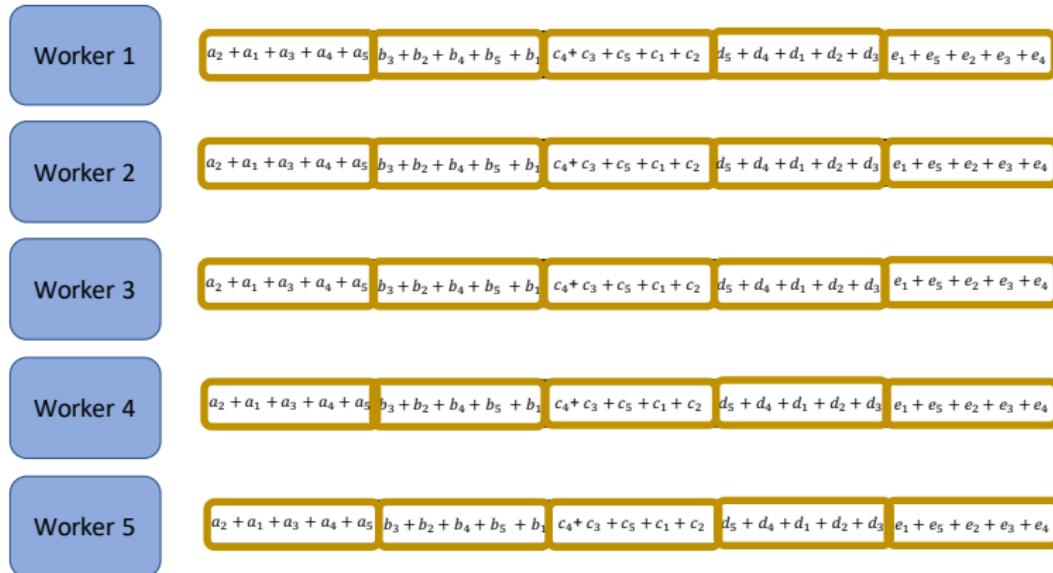
Summing arrays: allgather



Summing arrays: allgather



Summing arrays: allgather



Communication cost

- If there are N workers, each worker will send and receive values $N - 1$ times for scatter-reduce and $N - 1$ times for allgather.
- Each time, the workers will send K/N values, where K is the total number of values in the array that are being summed across the different workers.
- Then, the total amount of data transferred to and from every worker is

$$\text{data transferred} = 2(N - 1) \frac{K}{N},$$

which is independent of the number of workers.

Origins of Horovod

- ❑ The idea of using ring allreduce to efficiently train deep learning models was introduced by a team at Baidu in early 2017.
- ❑ Uber later adopted Baidu's draft implementation of the TensorFlow ring-allreduce algorithm and built upon it.
- ❑ The resulting Python package was named Horovod, "named after a traditional Russian folk dance in which performers dance with linked arms in a circle".



Horovod on Spark

- ❑ Two APIs provided: a high level **Estimator** API and a lower level **Run** API.
- ❑ The Estimator API is recommended if
 - training is done with Keras or PyTorch.
 - you want to train directly on a Spark Dataframe from `pyspark`.
 - you are using a standard gradient descent optimization process for training.
- ❑ The Run API offers more fine-grained control.



Distributed training framework
for TensorFlow, Keras, PyTorch,
and Apache MXNet.

Star 11,118

Navigation

[Overview](#)
[Concepts](#)
[Horovod Installation Guide](#)
[API](#)
[Horovod with TensorFlow](#)
[Horovod with Keras](#)
[Horovod with PyTorch](#)

Horovod on Spark

The `horovod.spark` package provides a convenient wrapper around Horovod that makes running distributed training jobs in Spark clusters easy.

In situations where training data originates from Spark, this enables a tight model design loop in which data processing, model training, and model evaluation are all done in Spark.

We provide two APIs for running Horovod on Spark: a high level **Estimator** API and a lower level **Run** API. Both use the same underlying mechanism to launch Horovod on Spark executors, but the Estimator API abstracts the data processing (from Spark DataFrames to deep learning datasets), model training loop, model checkpointing, metrics collection, and distributed training.

We recommend using Horovod Spark Estimators if you:

- Are using Keras (`tf.keras` or `keras`) or PyTorch for training.
- Want to train directly on a Spark DataFrame from `pyspark`.
- Are using a standard gradient descent optimization process as your training loop.

If for whatever reason the Estimator API does not meet your needs, the Run API offers more fine-grained control.

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

Horovod

Other technologies to watch out

References



[docs](#) [passing](#) [Ray](#) [Join Slack](#) [Discuss](#) [Ask Questions](#)

Ray provides a simple, universal API for building distributed applications.

Ray is packaged with the following libraries for accelerating machine learning workloads:

- [Tune](#): Scalable Hyperparameter Tuning
- [RLlib](#): Scalable Reinforcement Learning
- [Ray SGD](#): Distributed Training Wrappers
- [Ray Serve](#): Scalable and Programmable Serving

There are also many [community integrations](#) with Ray, including [Dask](#), [MARS](#), [Modin](#), [Horovod](#), [Hugging Face](#), [Scikit-learn](#), and others. Check out the [full list of Ray distributed libraries here](#).

Install Ray with: `pip install ray`. For nightly wheels, see the [Installation page](#).

Analytics Zoo



Distributed TensorFlow, PyTorch, Keras and BigDL on Apache Spark & Ray

Analytics Zoo is an open source **Big Data AI** platform, and includes the following features for scaling end-to-end AI to distributed Big Data:

- [Orca](#): seamlessly scale out TensorFlow and PyTorch for Big Data (using Spark & Ray)
- [RayOnSpark](#): run Ray programs directly on Big Data clusters
- [BigDL Extensions](#): high-level Spark ML pipeline and Keras-like APIs for BigDL
- [Zouwu](#): scalable time series analysis using AutoML

For more information, you may [read the docs](#).

<https://github.com/intel-analytics/analytics-zoo>

Contents

Neural networks for unstructured data

NNs in Spark ML

Deep learning with Spark

Ways of using neural networks/deep learning with Spark

Pandas UDFs

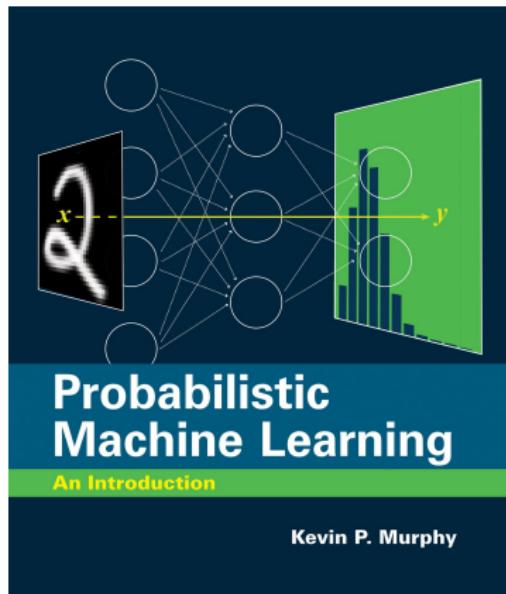
Horovod

Other technologies to watch out

References

References used in this lecture

Book: *Probabilistic Machine Learning: An introduction* by Kevin P Murphy,
MIT Press, 2022



<https://probml.github.io/pml-book/book1.html>

References used in this lecture

Website: *Spark Python API Documentation.*

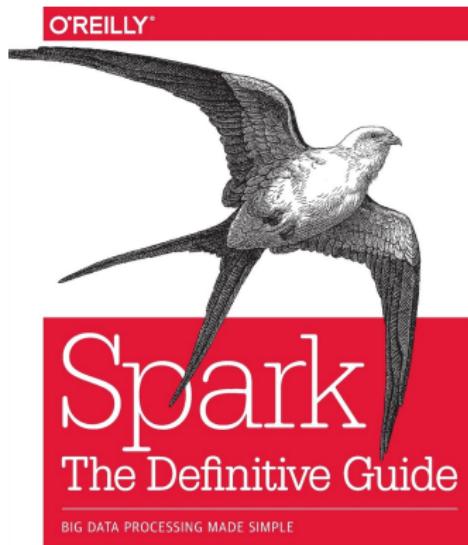
The screenshot shows the Apache Spark API Reference page. At the top, there's a navigation bar with links to 'Getting Started', 'User Guide', 'API Reference' (which is the active page), 'Development', and 'Migration Guide'. Below the navigation bar is a search bar labeled 'Search the docs ...'. To the left, there's a sidebar with a 'Spark' logo and a list of categories: 'Spark SQL', 'Pandas API on Spark', 'Structured Streaming', 'MLlib (DataFrame-based)', 'Spark Streaming', 'MLlib (RDD-based)', 'Spark Core', and 'Resource Management'. The main content area is titled 'API Reference' and contains the following text: 'This page lists an overview of all public PySpark modules, classes, functions and methods.' Below this, there's a hierarchical list of API sections:

- **Spark SQL**
 - Core Classes
 - Spark Session APIs
 - Configuration
 - Input and Output
 - DataFrame APIs
 - Column APIs
 - Data Types
 - Row
 - Functions
 - Window
 - Grouping
 - Catalog APIs
- **Pandas API on Spark**
 - Input/Output
 - General functions
 - Series
 - DataFrame
 - Index objects
 - Window
 - GroupBy
 - Machine Learning utilities
 - Extensions
- **Structured Streaming**
 - Core Classes
 - Input and Output
 - Query Management
- **MLlib (DataFrame-based)**
 - Pipeline APIs
 - Parameters
 - Feature

<https://spark.apache.org/docs/latest/api/python/reference/index.html>

References used in this lecture

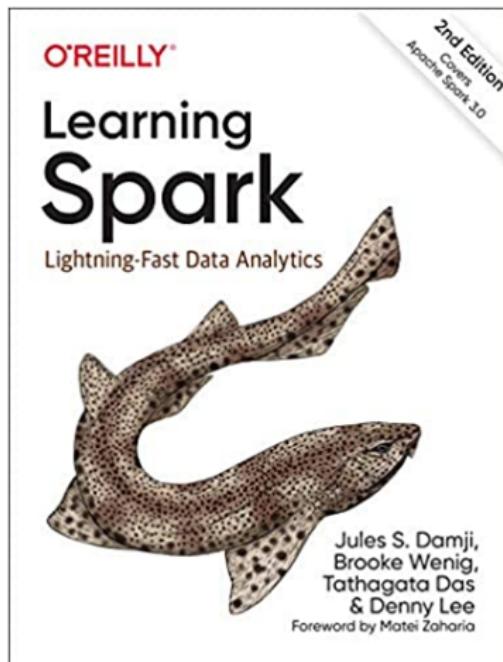
Book: *Spark: The Definitive Guide* by Bill Chambers & Matei Zaharia,
O'Reilly, 2018.



Bill Chambers & Matei Zaharia

References used in this lecture

Book: *Learning Spark: Lightning-Fast Data Analytics* by Jules Damji, Brooke Wenig, Tathagata & Denny Lee, O'Reilly, 2020.



References used in this lecture

Paper: *Horovod: fast and easy distributed deep learning in TensorFlow* by Alexander Sergeev and Mike Del Balso.

Horovod: fast and easy distributed deep learning in TensorFlow

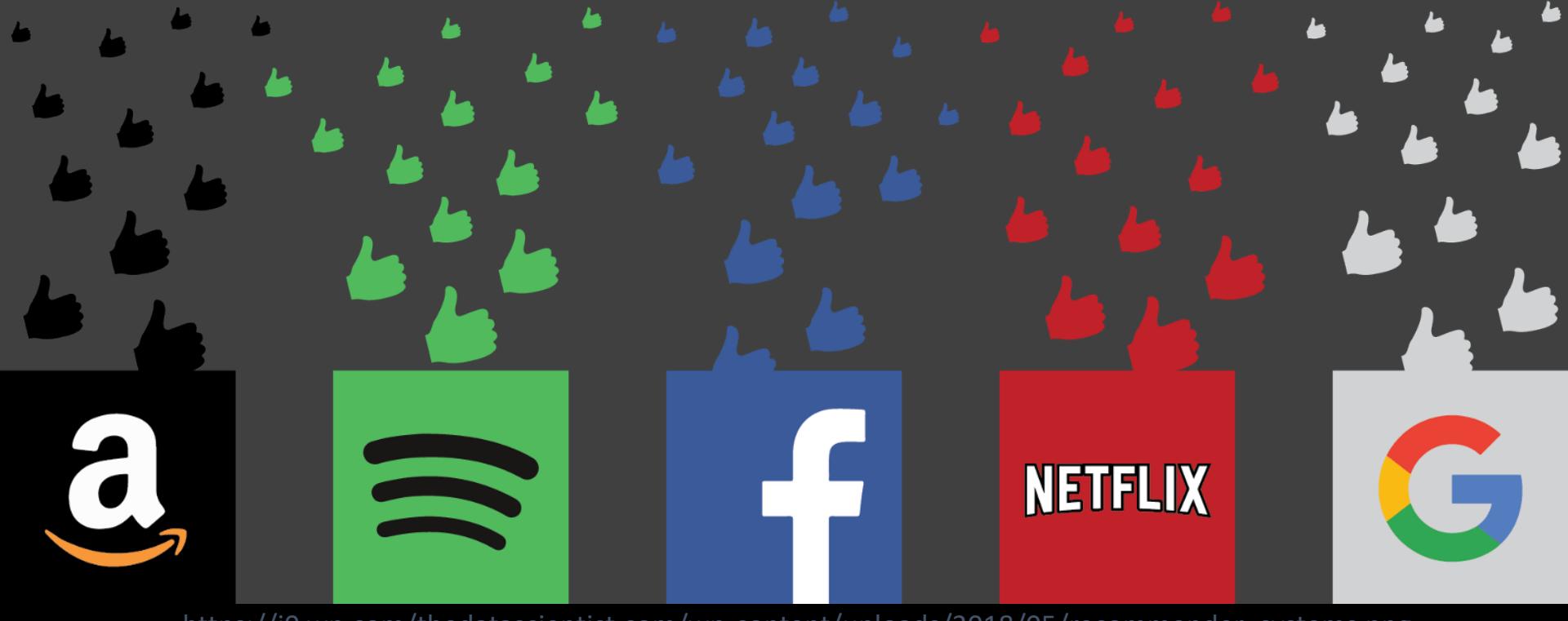
Alexander Sergeev
Uber Technologies, Inc.
asergeev@uber.com

Mike Del Balso
Uber Technologies, Inc.
mdb@uber.com

Abstract

Training modern deep learning models requires large amounts of computation, often provided by GPUs. Scaling computation from one GPU to many can enable much faster training and research progress but entails two complications. First, the training library must support inter-GPU communication. Depending on the particular methods employed, this communication may entail anywhere from negligible to significant overhead. Second, the user must modify his or her training code to take advantage of inter-GPU communication. Depending on the training library's API, the modification required may be either significant or minimal.

Existing methods for enabling multi-GPU training under the TensorFlow library entail non-negligible communication overhead and require users to heavily modify their model-building code, leading many researchers to avoid the whole mess and stick with slower single-GPU training. In this paper we introduce Horovod, an open source library that improves on both obstructions to scaling: it employs efficient inter-GPU communication via ring reduction and requires only a few lines of modification to user code, enabling faster, easier distributed training in TensorFlow. Horovod is available under the Apache 2.0 license at <https://github.com/uber/horovod>.



https://i0.wp.com/thedatascientist.com/wp-content/uploads/2018/05/recommender_systems.png

Lecture 7: Scalable Matrix Factorisation for Collaborative Filtering in RecSys

[COM6012: Scalable ML](#) by Haiping Lu

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/>

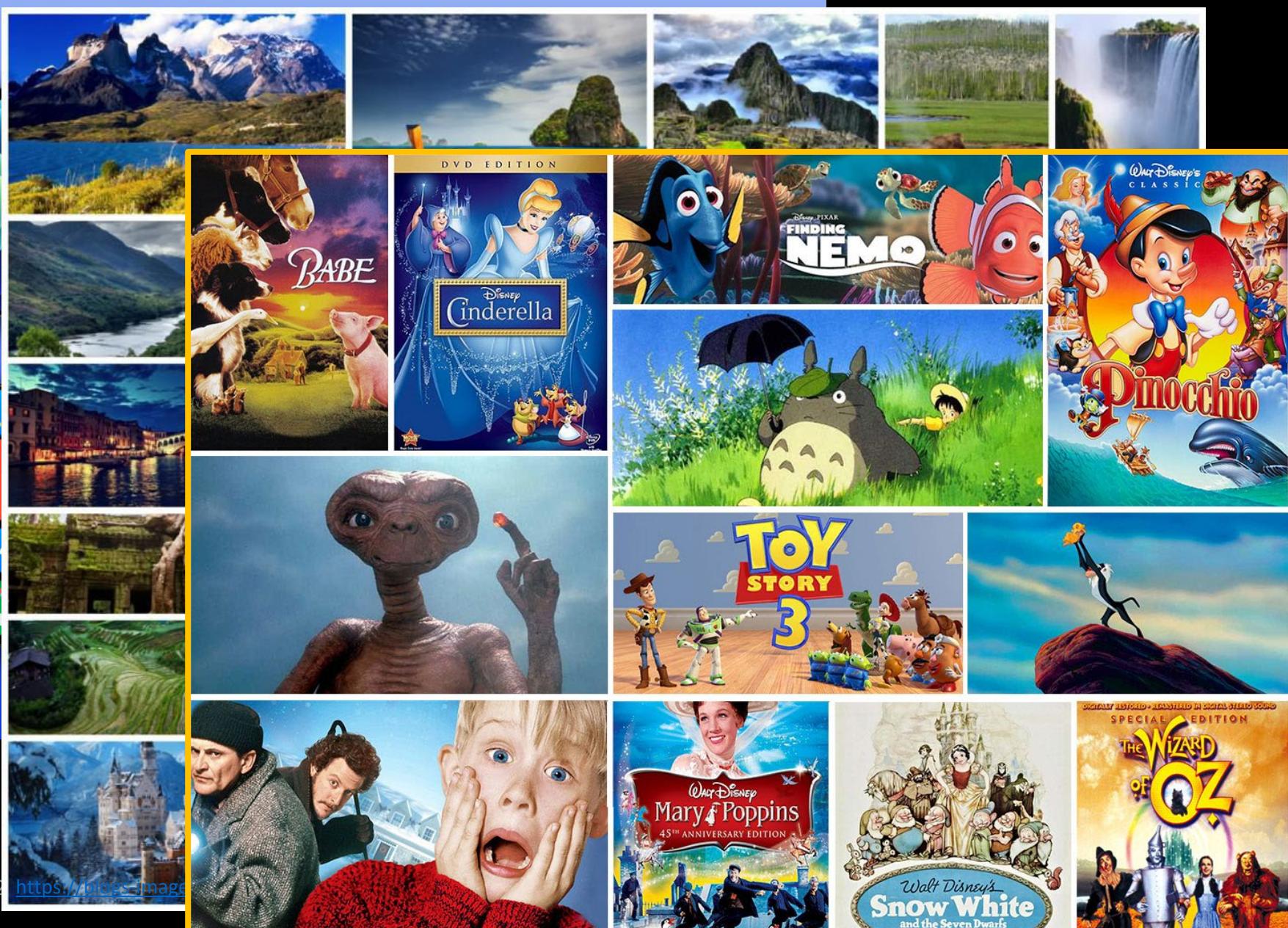
Week 7 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Week 7 Contents / Objectives

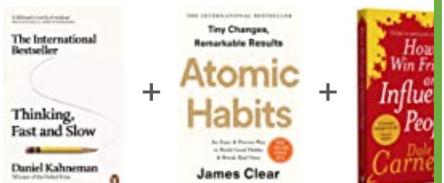
- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Many Decisions to Make



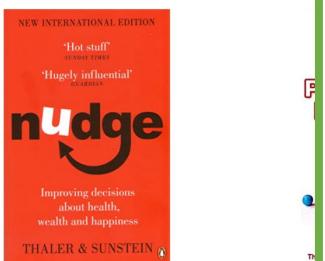
Recommendations Everywhere

Frequently bought together



- This item: Thinking, Fast and Slow by D...
- Atomic Habits: The life-changing millio...
- How to Win Friends and Influence Peop...

Customers who viewed this



Nudge: Improving Decisions About Health, Wealth and Happiness
Richard H. Thaler

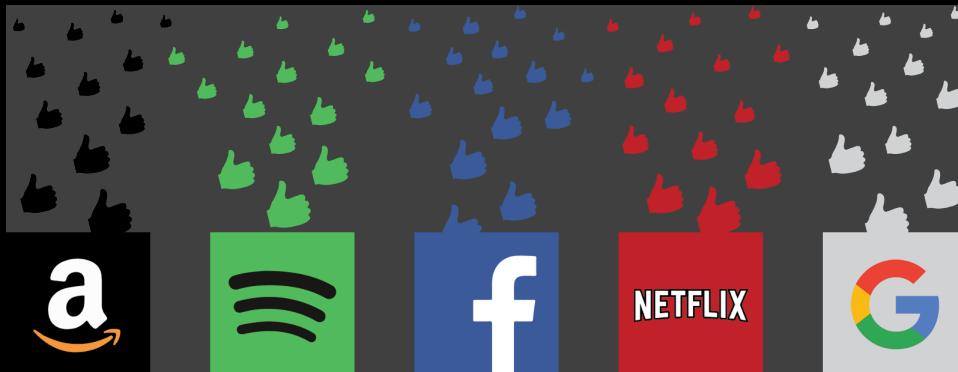
A screenshot of a search interface, likely Google Search, showing various recommendation cards. The interface includes a search bar with the word "job", a "Google Search" button, and a "Carbon foot print" footer.

The recommendations include:

- Online events for you**
 - Scholars Webinar on Drug Discovery...
Wed, Mar 24 - Thu, M...
Alireza Khaneymoori and 1,616 other...
[View](#)
 - Accelerating Visual Data Exploration...
Today, 12:30 PM
96 attendees
[View](#)
 - Live Chat with Salesforce Sr...
Tue, May 11, 5:30 PM
Vignesh Srinivasan and 733 other attendees
[View](#)
- People you may know from The University of Sheffield**
 - Twin Karma...
Research Software...
9 mutual connections
[Connect](#)
 - Andrew Stra...
Senior University Teacher...
19 mutual connections
[Connect](#)
 - Neil Walkins...
Senior Lecturer at The Universi...
30 mutual connections
[Connect](#)
 - Siobhan No...
Senior Lecturer at University of...
21 mutual connections
[Connect](#)

Recommender Systems (RecSys)

- Predict relevant items for a user, in a given context
- Predict to what extent these items are relevant
- A ranking task (searching as well)
- Implicit, targeted, intelligent advertisement
- Effective, popular marketing

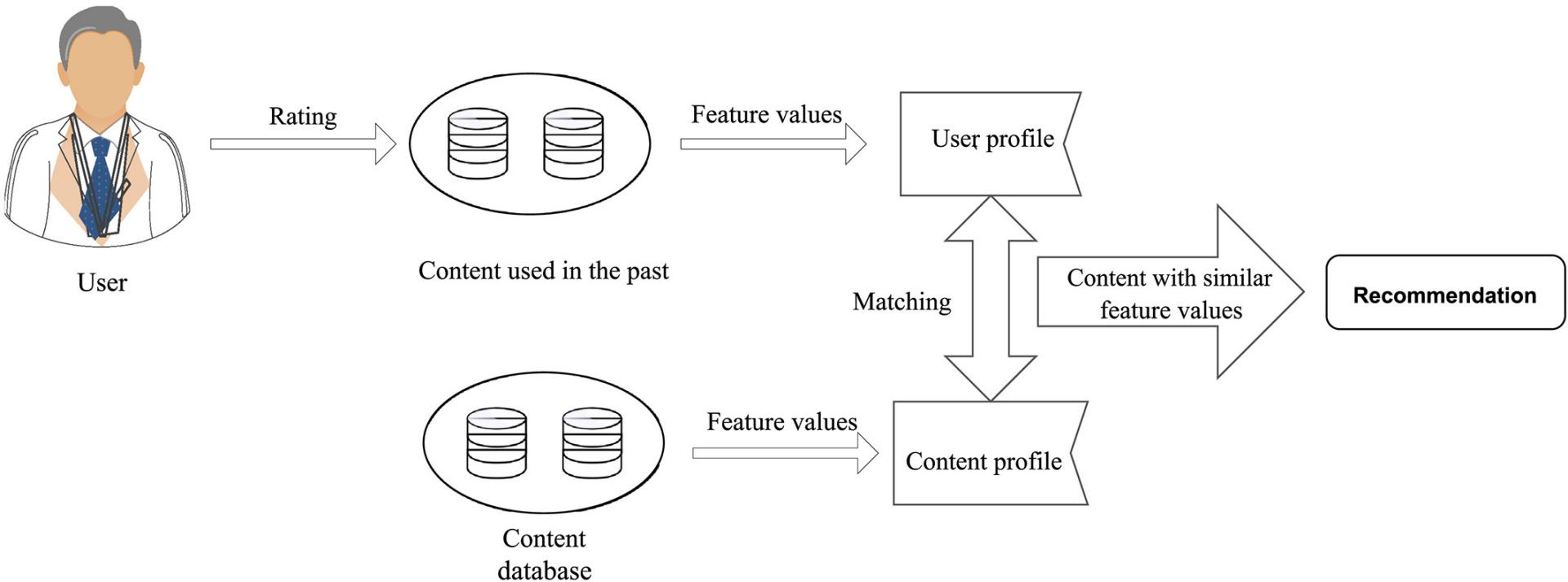


Two Classes of RecSys

- Content-based recommender systems
- Collaborative filtering recommender systems

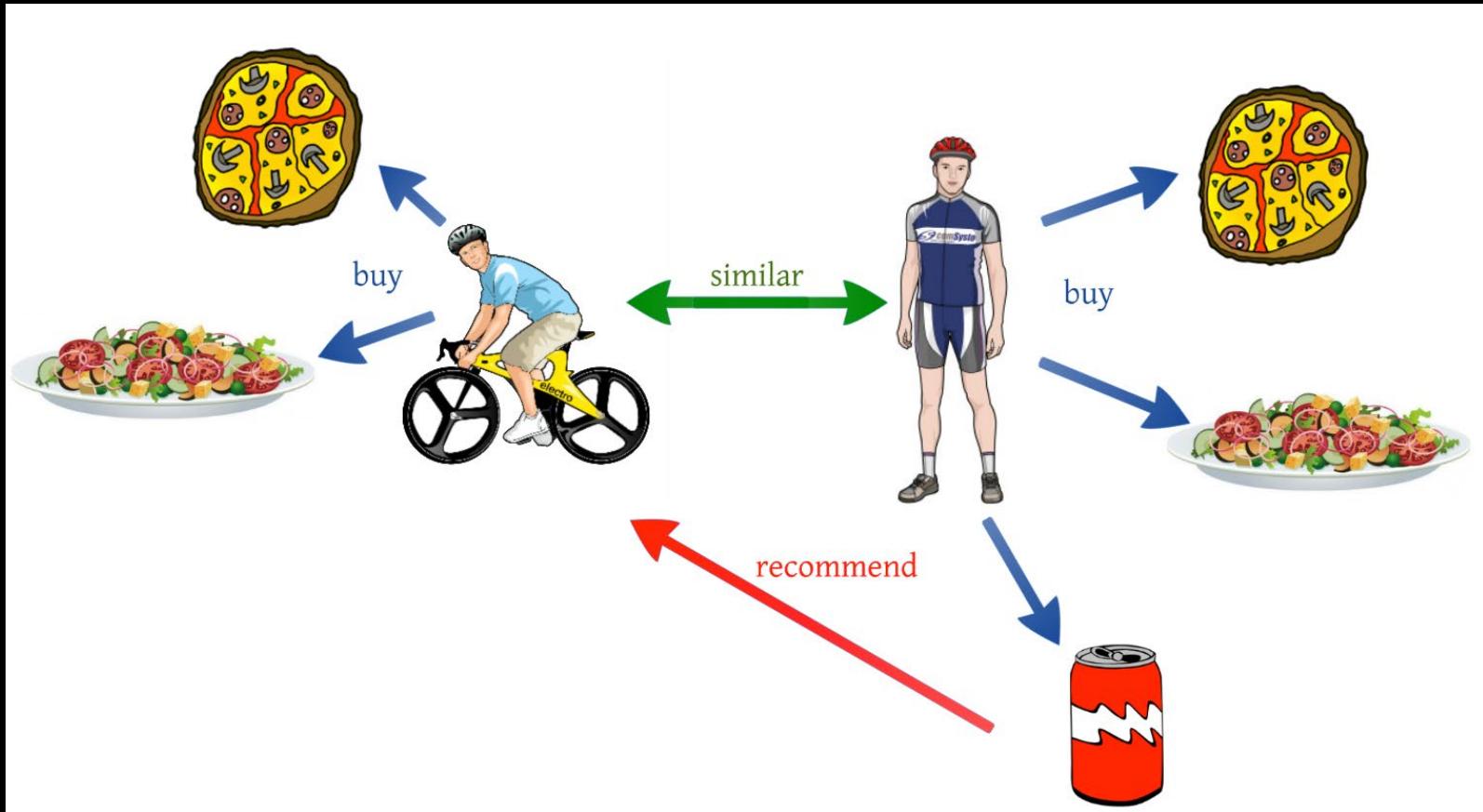


Content-based RecSys



[dac4058-fig-0010-m.jpg \(2128x789\) \(wiley.com\)](#)

Collaborative Filtering RecSys



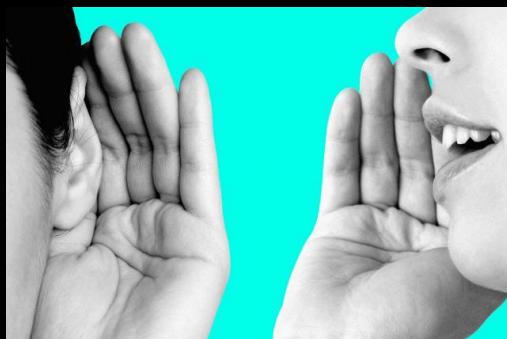
https://miro.medium.com/max/2656/1*6_NIX6CJYhtxzRM-t6ywkQ.png

Week 7 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

What is Collaborative Filtering?

- Information filtering based on past records
 - Electronic word of mouth marketing
 - Turn visitors into customers (e-Salesman)
- Components
 - Users (customers): who provide ratings
 - Items (products): to be rated
 - Ratings (interest): core data



John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

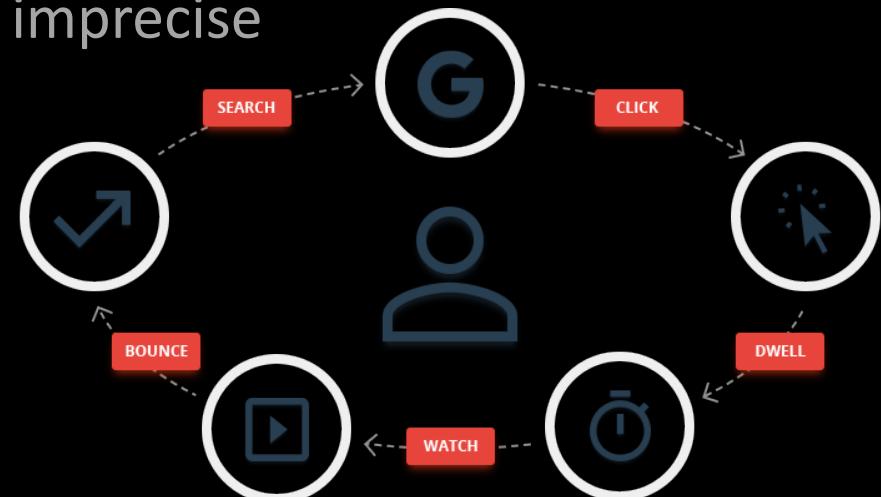
Collaborative Filtering (CF)

- Objective: predict how well a user will like an **unrated** item, given past ratings for a community of users
- How does CF work?
 - Input: many users' **ratings** for many items
 - Model: similar users \leftarrow ratings strongly correlate
 - Recommend items rated highly by similar users



Explicit vs Implicit Ratings

- Explicit (direct): users indicate levels of interest
 - Most accurate descriptions of a user's preference
 - Challenging in collecting data
- Implicit (indirect): observing user behavior
 - Can be collected with little or no cost to user
 - Ratings inference may be imprecise



Rating Scales

- Scalar ratings
 - Numerical scales
 - 1-5, 1-7, etc.
- Binary ratings
 - Agree/Disagree, Good/Bad, etc.
- Unary ratings
 - Presence/absence of an event, e.g., purchase/browsing history, search patterns, mouse movements
 - No info about the opposite $\neq 0$



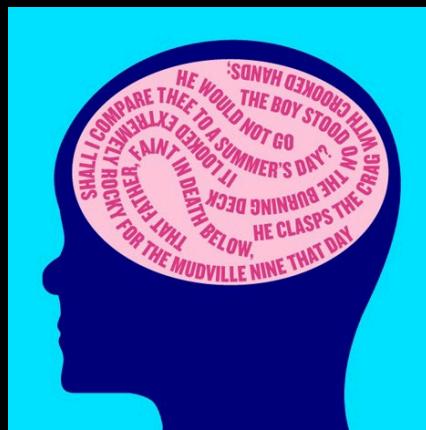
CF Preferences

- Many users, many items, many ratings
- Users rate multiple items
- Other users with similar needs/tastes
- Item evaluation requires personal taste
- Taste persists



CF Methods

- Memory-based: predict using past ratings **directly**
 - Weighted ratings given by other similar users
 - User-based & item-based (non-ML)
- Model-based: **model** users based on past ratings
 - Predict ratings using the learned model



[iforget-465.jpg \(465x465\) \(newyorker.com\)](#)



[neural-header.jpg \(756x503\) \(utsouthwestern.edu\)](#)

Prediction Accuracy

- Mean absolute error (MAE)

$$MAE = \frac{\sum_{i,j} |p_{i,j} - r_{i,j}|}{n}$$

- Normalized MAE

$$NMAE = \frac{MAE}{r_{max} - r_{min}}$$

- Root mean squared error (RMSE)

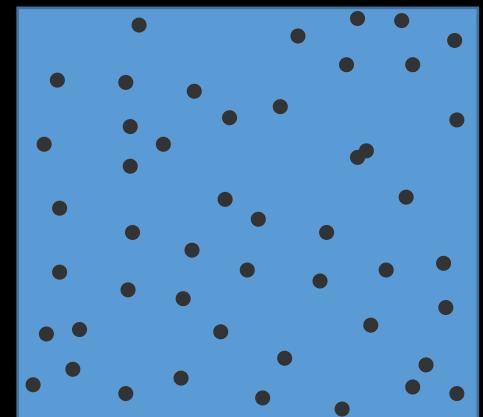
$$RMSE = \sqrt{\frac{1}{n} \sum_{i,j} (p_{i,j} - r_{i,j})^2}$$

Challenges

- **Cold Start**
 - New user
 - Rate some initial items
 - Non-personalized rec.
 - Describe tastes
 - Demographic info
 - New item
 - Randomly selecting items
 - Content analysis, metadata (non-CF)
- **Sparsity:** sparse user-item matrix
- **Scalability:** millions of users and items



[isbil+2.jpg \(490x303\) \(squarespace-cdn.com\)](#)



Week 7 Contents / Objectives

- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Matrix Factorisation (MF) for CF

- Characterise items/users by vectors of factors learned from the rating matrix **user x item**
- High correlation between item and user factors → good recommendation
- Flexibility: incorporate implicit feedback, temporal effects, and confidence levels

John	5	1	3	5
Tom	?	?	?	2
Alice	4	?	3	?

Basic MF Model

- Map users & items to a **joint latent factor** space of dimensionality k
 - Item $i \rightarrow$ vector q_i : the extent to which the item possesses those k factors
 - User u : vector p_u : the extent of interest the user has on those k factors
- User-item interactions: the user's overall interest in the item's characteristics
 - Inner product $q_i^T p_u$: predicted user u 's rating of item i

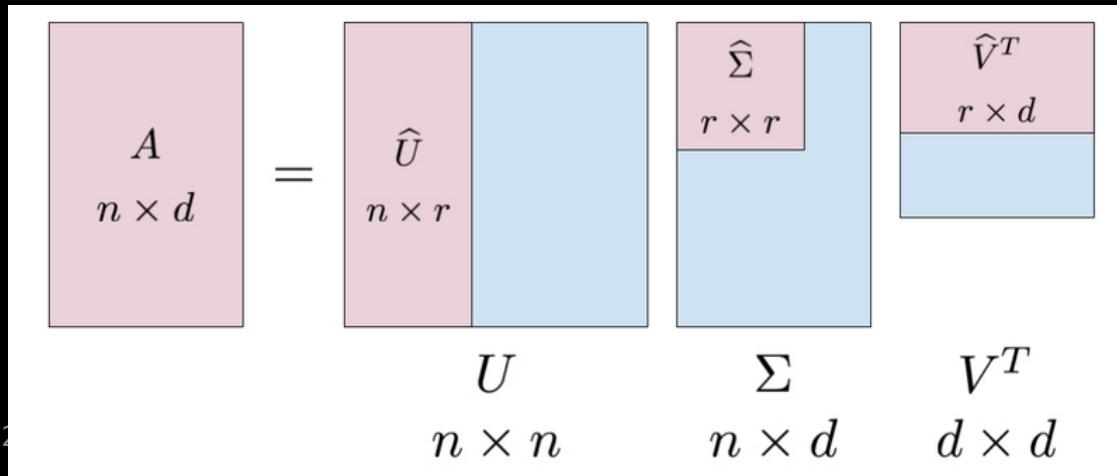
$$\hat{r}_{u,i} = q_i^T p_u$$

How to Learn the MF Model

- To learn: item factors $\{q_i\}$ and user factors $\{p_u\}$
- Factorisation assuming full rating matrix
 - Factorise rating matrix R using SVD to obtain P, S, Q

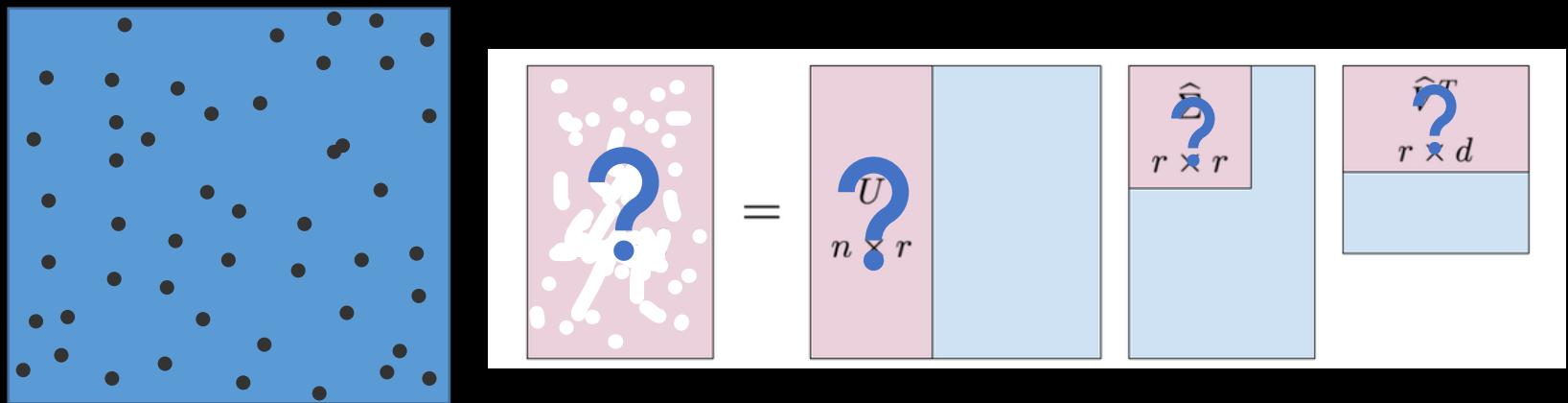
$$R = PSQ^T$$

- Reduce the matrix S to dimension k , i.e. S_k
- $P \rightarrow P_k$ and $Q \rightarrow Q_k$: $P_k S_k \rightarrow \hat{P}$, and $S_k Q_k^T \rightarrow \hat{Q}^T$
- u th row of $\hat{P} \rightarrow p_u$, i th column of $\hat{Q}^T \rightarrow q_i$



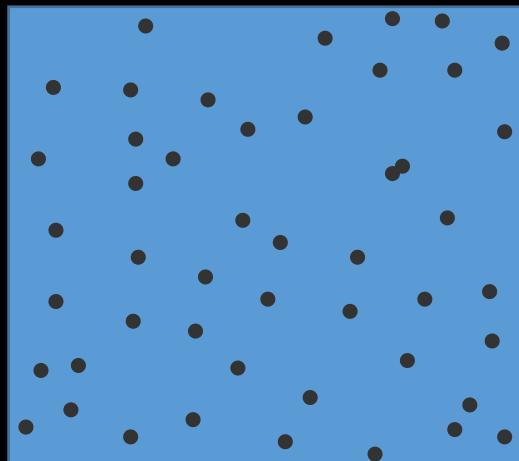
Challenges in MF for CF

- High portion of missing values caused by sparseness in the user-item rating matrix
- Conventional SVD is undefined when knowledge about the matrix is incomplete



How to Fill Missing Values

- Imputation: fill in missing ratings using the average ratings for user and item
- Problems
 - Expensive: significantly increases the amount of data
 - Inaccurate imputation might distort the data



MF with Missing Values

- Modelling directly the **observed ratings only**
 - Avoid overfitting through a regularised model
 - Minimize the regularised squared error on the set of known ratings to learn the factor vectors p_u and q_i

$$\min_{q^*, p^*} \sum_{(u,i) \in \kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|_2^2 + \| p_u \|_2^2)$$

- κ : the training set of the (u, i) pairs with known ratings
- λ : the regularisation parameter

Alternating Least Squares for MF-CF

$$\min_{q^*, p^*} \sum_{(u,i) \in \kappa} (r_{ui} - q_i^T p_u)^2 + \lambda(\| q_i \|_2^2 + \| p_u \|_2^2)$$

- Both p_u and q_i are unknown (non-convex function)
- Fix one of them → quadratic with optimal solution
- Alternating Least Squares (ALS): alternate between fixing q_i s and fixing p_u s
 - Fix $P(p_u)$ s as \hat{P} to recompute q_i s by solving a least-squares problem $\| R - PQ^T \|_F$ (Frobenius norm)

$$R = \hat{P}Q^T \Rightarrow Q^T = (\hat{P}^T \hat{P})^{-1} \hat{P}^T R$$

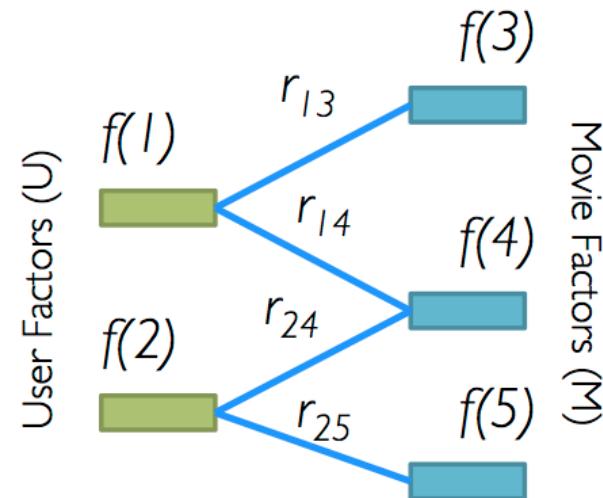
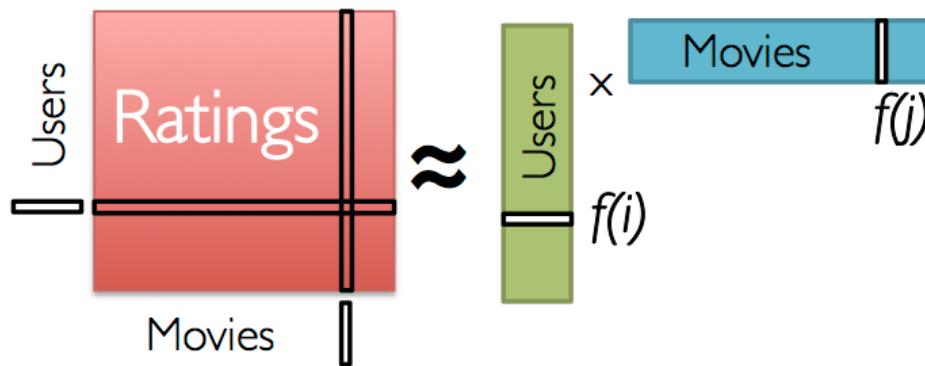
- Fix Q as \hat{Q} , we have

$$P = R\hat{Q}(\hat{Q}^T \hat{Q})^{-1}$$

- Random initialisation to start this iteration

MF for Movie Recommendation

Low-Rank Matrix Factorization:



Iterate:

$$f[i] = \arg \min_{w \in \mathbb{R}^d} \sum_{j \in \text{Nbrs}(i)} (r_{ij} - w^T f[j])^2 + \lambda ||w||_2^2$$

[spark-training/matrix_factorization.png at master · databricks/spark-training \(github.com\)](https://github.com/databricks/spark-training/blob/master/spark-training/matrix_factorization.png)

Week 7 Contents / Objectives

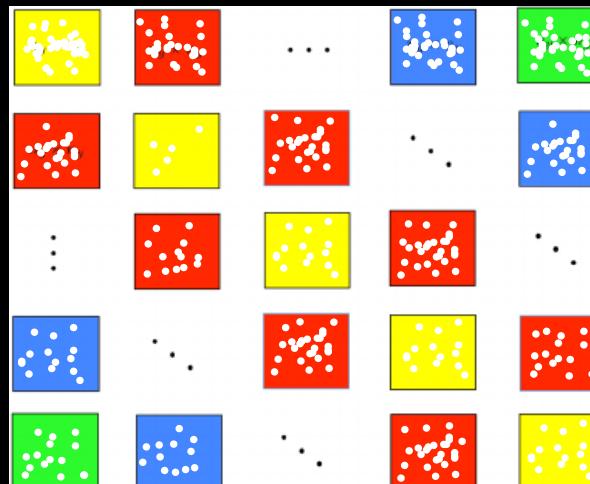
- Recommender Systems
- Collaborative Filtering
- Matrix Factorisation for Collaborative Filtering
- Scalable Collaborative Filtering in Spark

Key in Scalable ML

- Computation and storage should be **linear** (in n, d)
→ Low-cost computation (time + space)
- Perform **parallel** and **in-memory** computation
→ Many working + reduce disk I/O
- Minimise network **communication** → Reduce overhead in parallelisation, not the more the better

Blocked Implementation of ALS

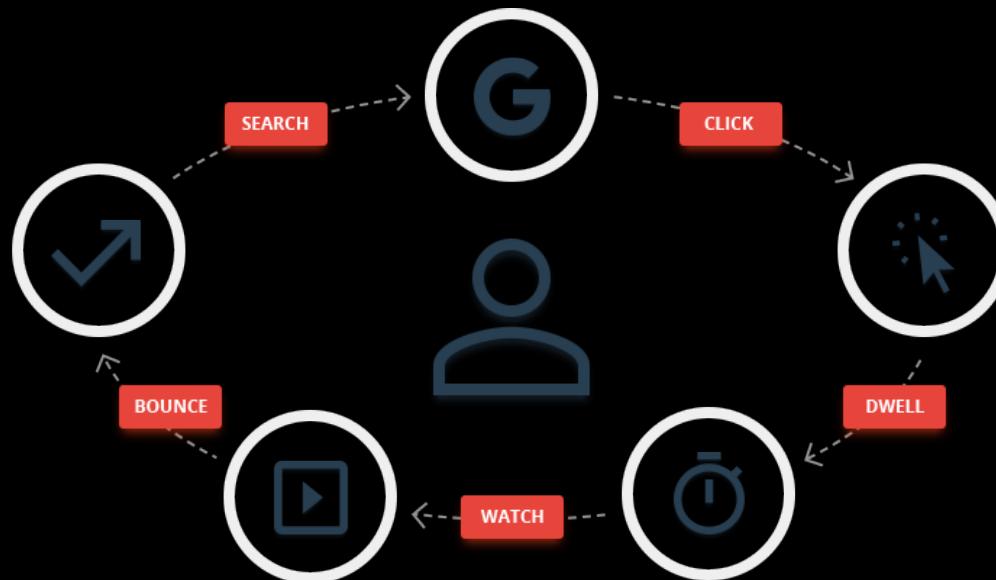
- Group users and items into blocks
 - Reduce **communication**: only send one copy of each user vector to each item block on each iteration, and only for the item blocks that need that user's feature vector
 - Pre-compute info: **out-links** of each user (which blocks of items it will contribute to); **in-links** for each item (which of the feature vectors it receives from each user block it will depend on)



[Color-online-A-symmetric-block-Toeplitz-matrix-Each-block-is-also-a-symmetric-Toeplitz.png \(488x369\) \(researchgate.net\)](#)

Implicit Feedback Modelling

- Implicit feedback: views, clicks, purchases, likes, shares
 - Rating r = strength in observations of user actions (#clicks, viewing duration) → confidence level in observed user preference
 - Construct a preference matrix P : e.g. 1 if $r > 0$ and 0 if $r = 0$
 - Factorisation of P → latent factors to predict the preference of a user for an item (details in an [ICDM08 paper](#))



The ALS API in Spark

- **numUserBlocks/numItemBlocks**: the number of blocks the users/items will be partitioned into to parallelize computation (defaults to 10)
- **rank**: the number of latent factors in the model (defaults to 10)
- **regParam**: the regularization parameter in ALS (defaults to 0.1)
- **implicitPrefs**: whether to use the explicit feedback ALS variant or one adapted for implicit feedback data (defaults to false: explicit ratings)
- **alpha**: the baseline confidence of implicit feedback (defaults to 1.0)
- **nonnegative**: whether to use nonnegative constraints (defaults to false)
- **coldStartStrategy**: “drop” → drop any rows in the DataFrame of predictions that contain NaN values (defaults to “nan”: assign NaN to a user and/or item factor is not present in the model)
- **blockSize**: the size of the user/product blocks in the blocked implementation of ALS to reduce communication

```

944     def train[ID: ClassTag]( // scalastyle:ignore
945         ratings: RDD[Rating[ID]],
946         rank: Int = 10,
947         numUserBlocks: Int = 10,
948         numItemBlocks: Int = 10,
949         maxIter: Int = 10,
950         regParam: Double = 0.1,
951         implicitPrefs: Boolean = false,
952         alpha: Double = 1.0,
953         nonnegative: Boolean = false,
954         intermediateRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
955         finalRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
956         checkpointInterval: Int = 10,
957         seed: Long = 0L)(
958         implicit ord: Ordering[ID]): (RDD[(ID, Array[Float])], RDD[(ID, Array[Float])]) = {
959
960         require(!ratings.isEmpty(), s"No ratings available from $ratings")
961         require(intermediateRDDStorageLevel != StorageLevel.NONE,
962                 "ALS is not designed to run without persisting intermediate RDDs.")
963
964         val sc = ratings.sparkContext
965
966         // Precompute the rating dependencies of each partition
967         val userPart = new ALSPartitioner(numUserBlocks)
968         val itemPart = new ALSPartitioner(numItemBlocks)
969         val blockRatings = partitionRatings(ratings, userPart, itemPart)
970             .persist(intermediateRDDStorageLevel)
971         val (userInBlocks, userOutBlocks) =
972             makeBlocks("user", blockRatings, userPart, itemPart, intermediateRDDStorageLevel)
973         userOutBlocks.count()    // materialize blockRatings and user blocks

```

CF in Spark ML

- [Scala code](#) (1800+ lines)
- Documentation: [Collaborative Filtering in Spark](#)
- [DataBricks movie recommendations tutorial](#)
- [DataBricks](#): founded by the creators of Apache Spark
 - Their latest packages at [their GitHub page](#)
 - Databricks community edition: 15GB memory **free** (?)



References

- Yehuda Koren, Robert Bell, and Chris Volinsky. "Matrix factorization techniques for recommender systems." *Computer* 8 (2009): 30-37 (Yahoo & AT&T)
- Yifan Hu, Yehuda Koren, and Chris Volinsky. "Collaborative filtering for implicit feedback datasets." *Eighth IEEE International Conference on Data Mining*, 2008
- Charu C. Aggarwal, Recommender Systems: The Textbook, April 2016



https://miro.medium.com/max/6000/1*wJZbLHQ7kgyh92dtkj9B6Q.png

Lecture 8: Scalable k -means Clustering

[COM6012: Scalable ML](#) by [Haiping Lu](#)

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/>

Week 8 Contents / Objectives

- Introduction to Cluster Analysis
- k -means Clustering
- Scalable k -means
- k -means in Spark & Limitations

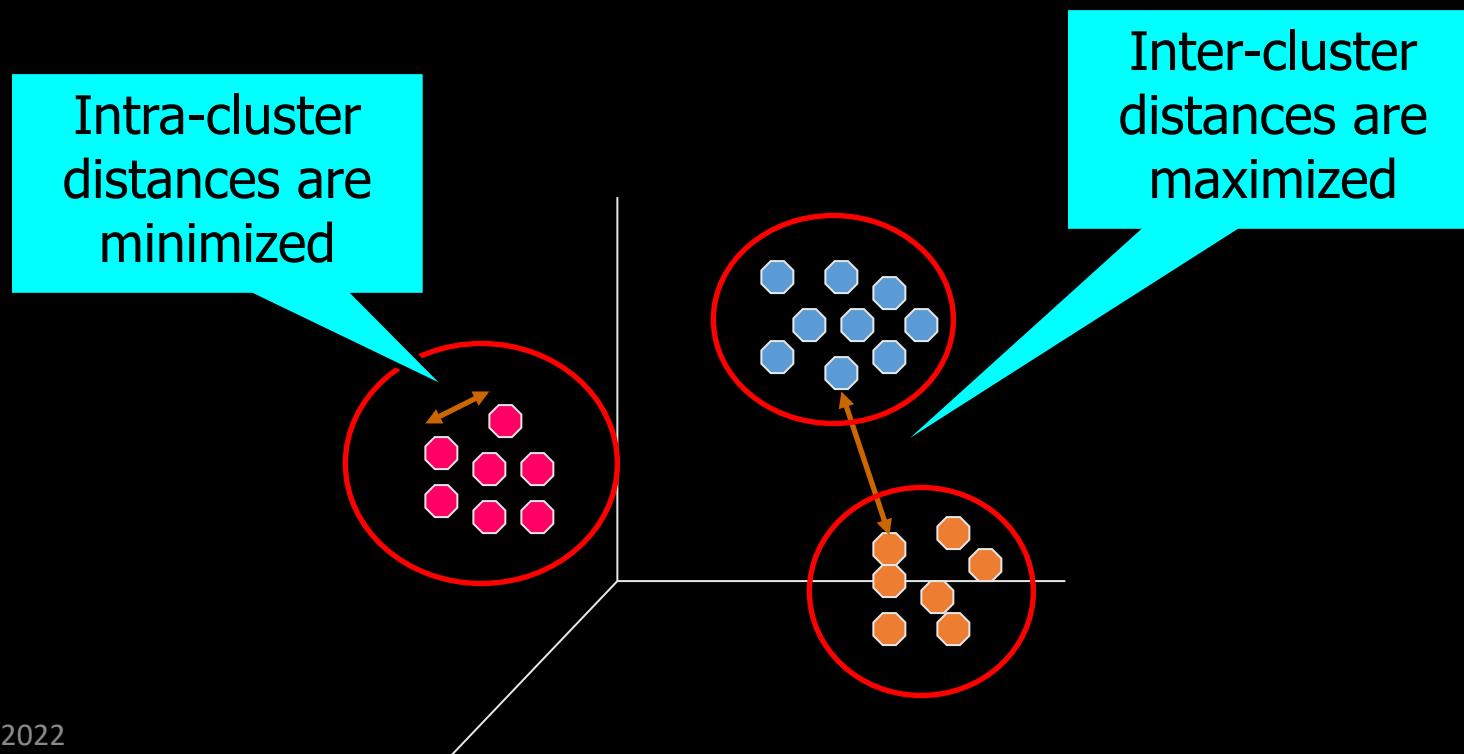
Week 8 Contents / Objectives

- Introduction to Cluster Analysis
- k -means Clustering
- Scalable k -means
- k -means in Spark & Limitations

Cluster Analysis

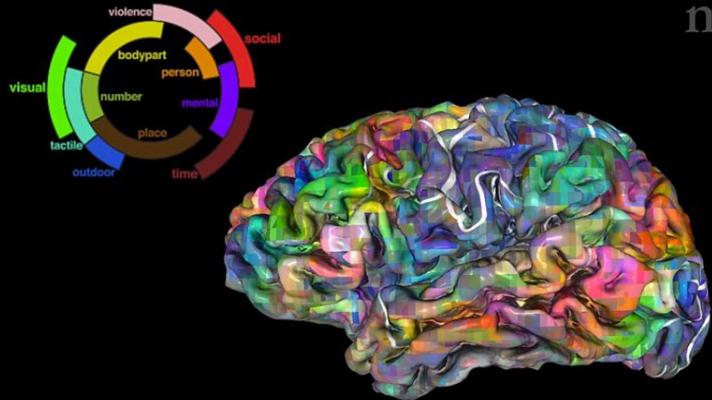
Output	Supervised	Unsupervised
Discrete	Classification	Clustering
Continuous	Regression	Dimensionality Reduction

- “Classification without labelled data”
- “Dimensionality reduction to 1 (cluster index)”



Cluster Analysis

- Divide data into clusters that are meaningful/useful
- Clusters: pseudo-classes
- Key for exploratory data analysis in many areas
 - Brain parcellation, social network analysis, customer segmentation, patient stratification, drug discovery, ...



[brain-video2.jpg \(1014×570\) \(wp.com\)](#)



https://miro.medium.com/max/1145/1*YjvoSe-hGaPpbWi5fDqV2w.png



[segmentation3.png \(586×236\) \(visionedgemarketing.com\)](#)

Anomaly/Outlier Detection

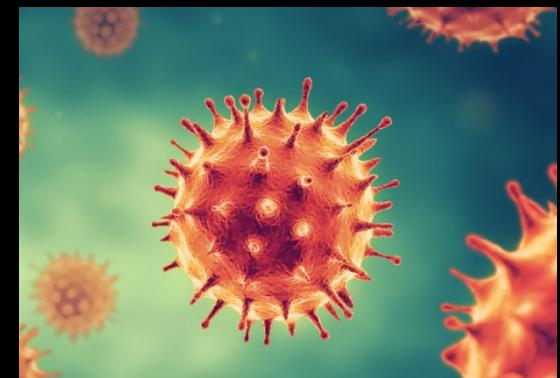
- Anomalies: data points very different from others
- Applications: credit card fraud, network intrusion, unknown virus, eco disturbance, unusual symptom, ...



[financial_protection_credit_card_fraud_img1.jpg \(512×347\) \(traderdefenseadvisory.com\)](#)



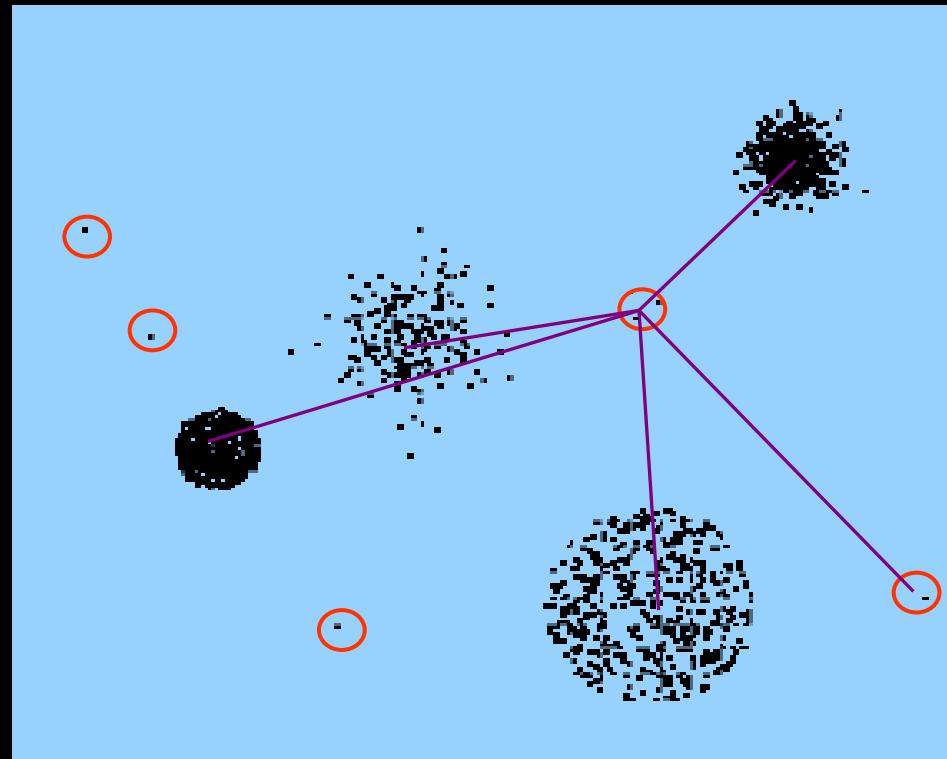
[Network-Intrusion-Detection-and-Prevention-.jpg \(1600×962\) \(wp.com\)](#)



[newseventsimage_1613554577167_mainnews2012_x1.jpg \(700×484\) \(imperial.ac.uk\)](#)

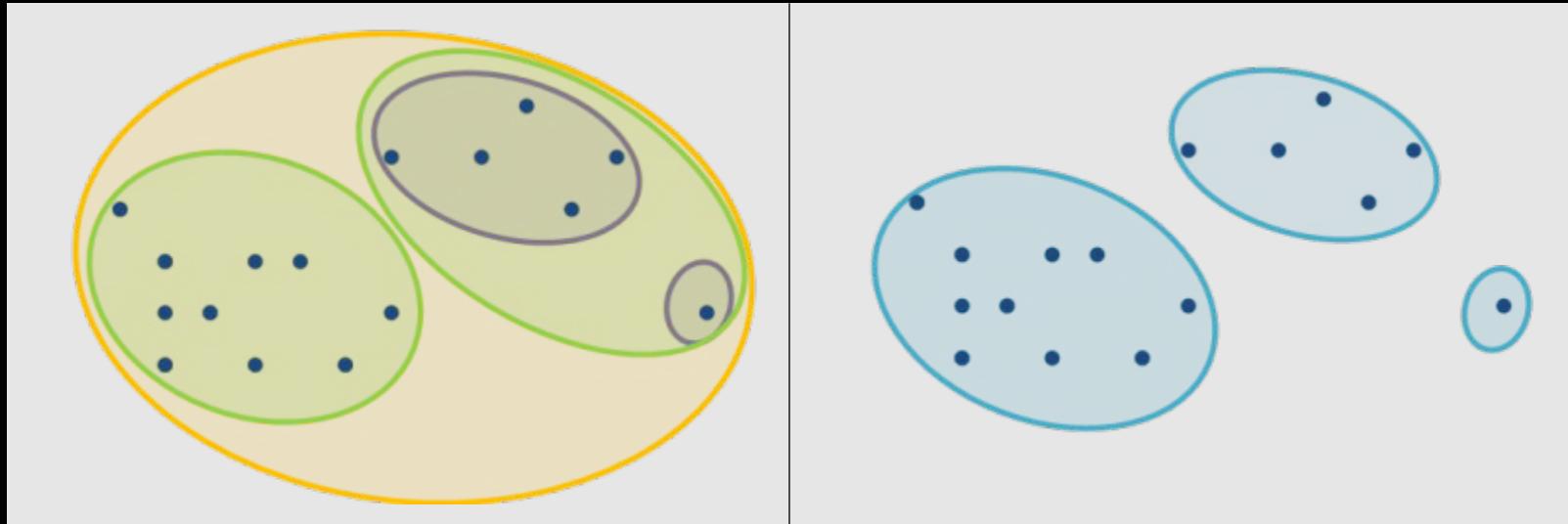
Clustering-based Anomaly Detection

- Cluster the data points
- Those in small clusters
→ candidate outliers
- Compute the distance between candidate points and non-candidate clusters
- Candidate points far from all other non-candidate points → outliers

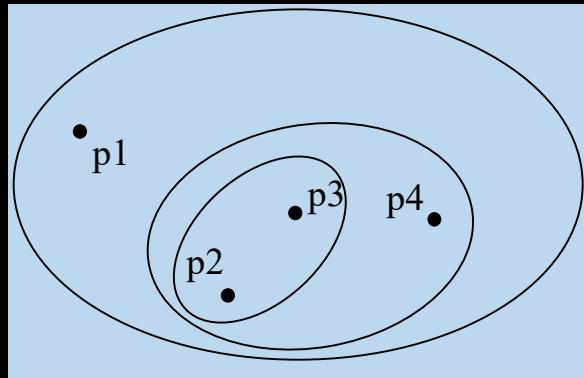


Hierarchical vs Partitional Clustering

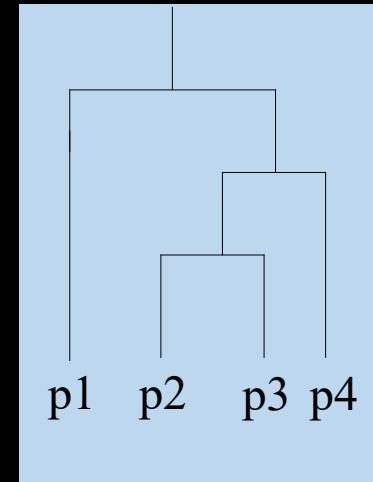
- Hierarchical: **nested** clusters as a hierarchical tree
 - Each node (cluster) in the tree (except for the leaf nodes) is the union of its children (subclusters)
 - The root of the tree → the cluster containing all data points
- Partitional: **non-overlapping** clusters
 - Each data point is in exactly one cluster



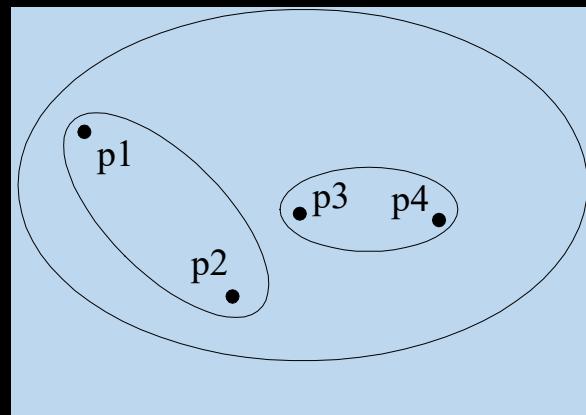
Hierarchical Clustering



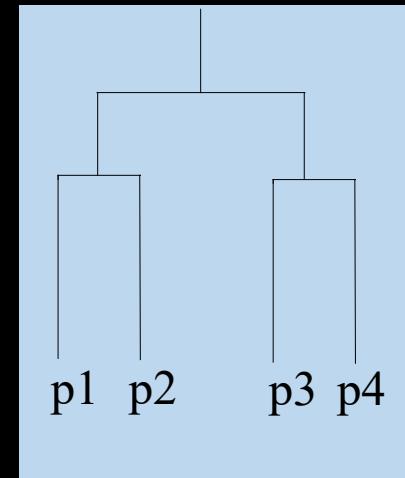
Traditional Hierarchical Clustering



Traditional Dendrogram



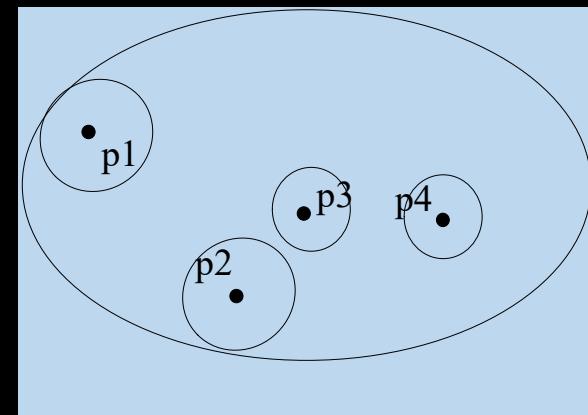
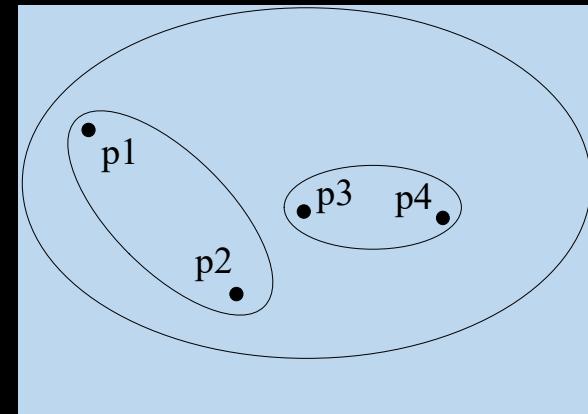
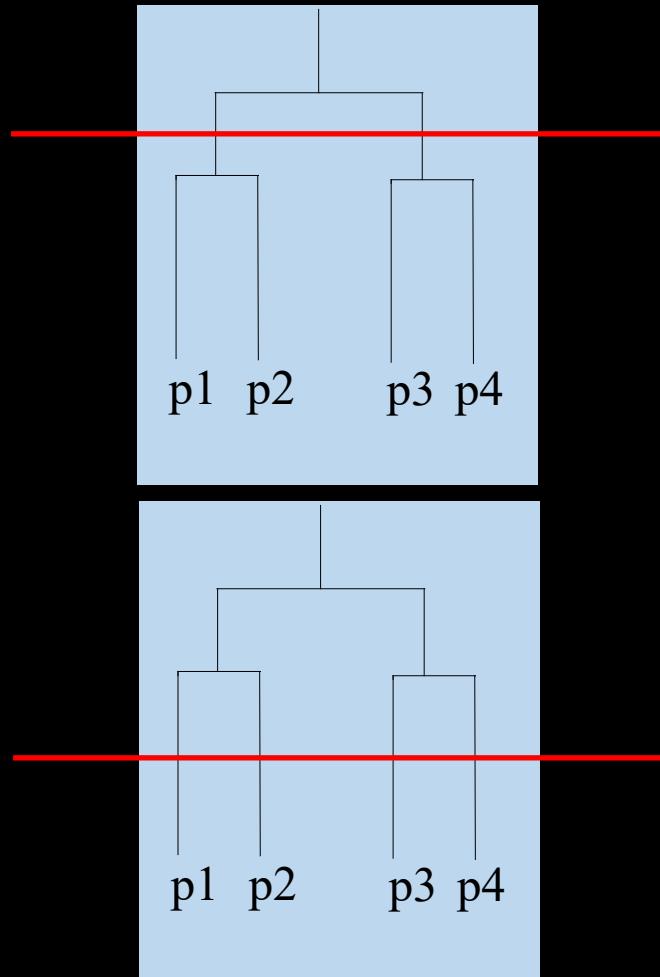
Non-traditional Hierarchical Clustering



Non-traditional Dendrogram

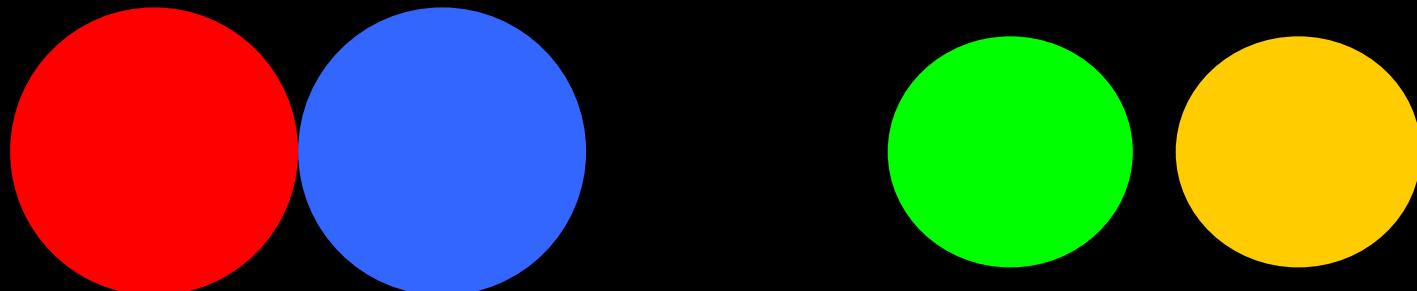
Hierarchical → Partitional

- Hierarchical = a sequence of partitional clustering cutting the hierarchical tree at a particular level



Centre/Prototype-based Clusters

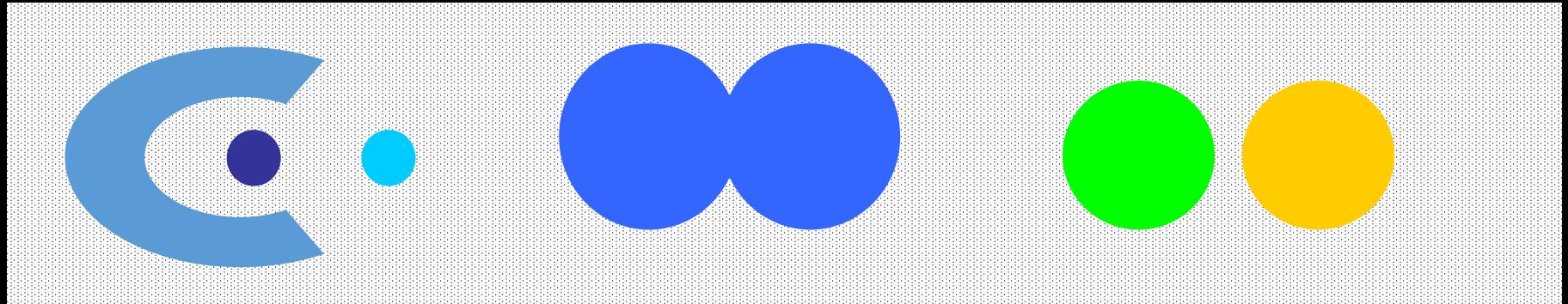
- Data points in a cluster are closer (more similar) to the centre of that cluster than to the centre of any others
- Centre
 - Centroid: the **average** of all the points in a cluster
 - Medoid: the most **representative** point of a cluster (e.g., categorical)



4 centre-based clusters

Density-based Clusters

- Cluster: a dense region of points separated by low-density regions from other regions of high density
- Used when the clusters are irregular/intertwined, and when noise and outliers are present



6 density-based clusters

Week 8 Contents / Objectives

- Introduction to Cluster Analysis
- *k*-means Clustering
- Scalable *k*-means
- *k*-means in Spark & Limitations

k -means Clustering

- A **centre-based, partitional** clustering approach
- Input: a set of n data points $X = \{x_1, x_2, \dots, x_n\}$ and the number of clusters k
- For a set $C = \{c_1, c_2, \dots, c_k\}$ of cluster centres, define the **Sum of Squared Error (SSE)** as:

$$SSE_X(C) = \sum_{x \in X} d(x, C)^2$$

$d(x, C)$: distance from x to the closest centre in C

- Goal: find C centres minimising $SSE_X(C)$

Lloyd Algorithm for k -means

- Start with k centres $\{c_1, c_2, \dots, c_k\}$ chosen uniformly at random from data points
- Assign clusters and compute centroids till convergence
 - Alternating optimisation again, similar to ALS
- Limitations
 - Many iterations to converge
 - Sensitive to initialisation
 - Random initialisation can get two centres in the same cluster → stuck in a local optimum (example on next slide)

Initialisation → Stuck

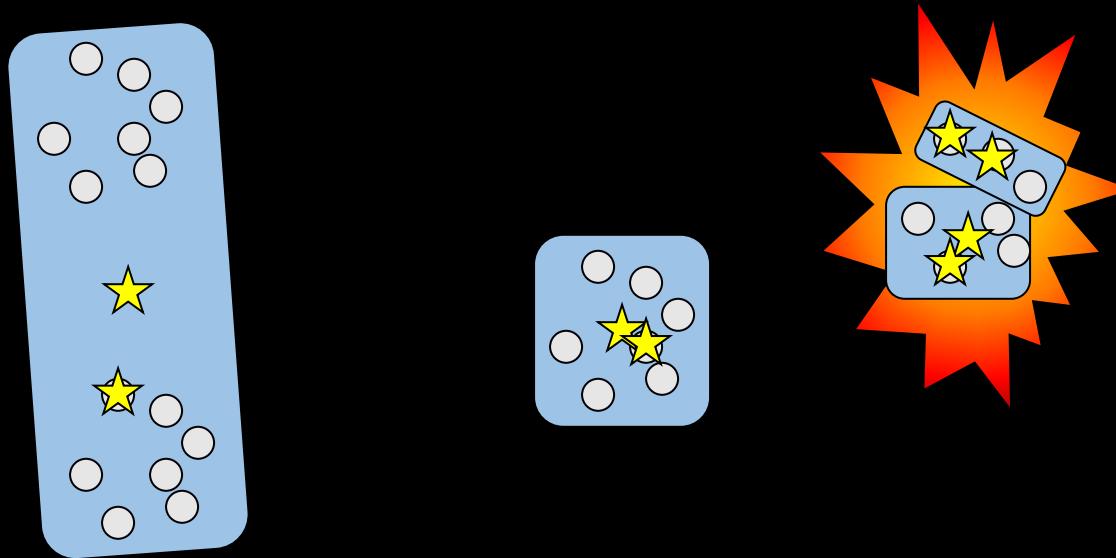


Figure credited to David Arthur

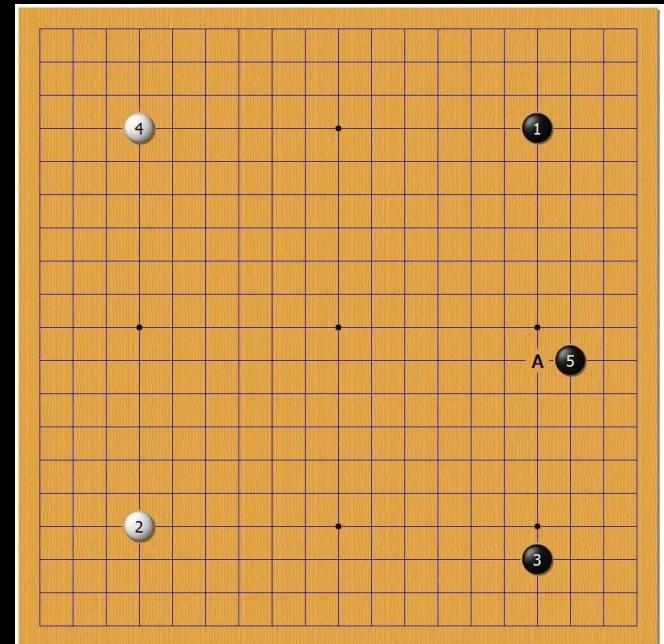
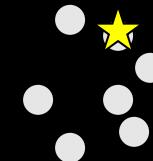
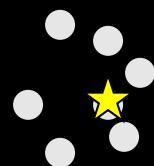
Week 8 Contents / Objectives

- Introduction to Cluster Analysis
- k -means Clustering
- Scalable k -means
- k -means in Spark & Limitations

k -means++ [Arthur et al. '07]

- Key idea: spread out the centres
- Choose the first centre c_1 uniformly at random
- Repeat for $2 \leq i \leq k$:
 - Choose c_i to be equal to a data point x_0 sampled from the distribution:
$$\frac{d(x_0, C)^2}{SSE_X(C)} \propto d(x_0, C)^2$$
 - Reminder: $d(x, C) = \text{distance from } x \text{ to the closest centre in } C$
- Theorem: $O(\log k)$ -approximation to the optimum

k -means++ Initialisation



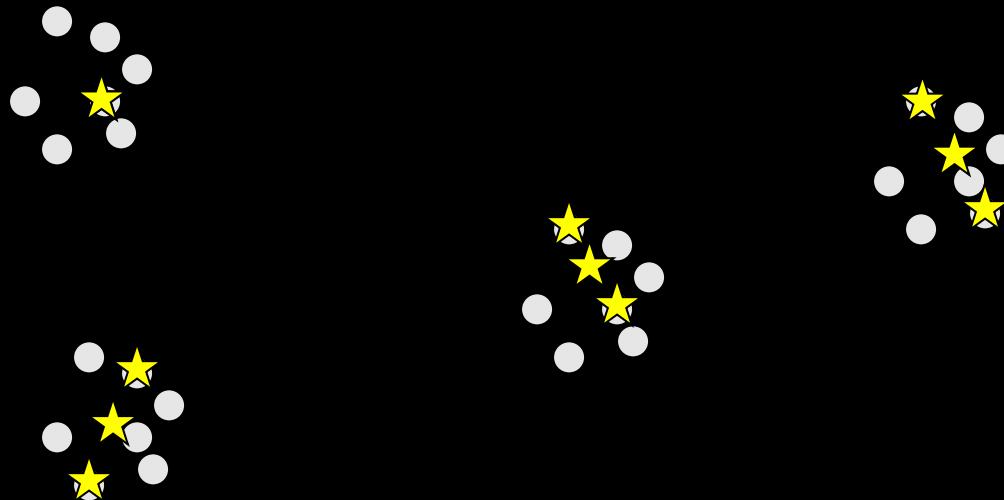
k -means++ \rightarrow k -means||

- k -means++ limitations
 - Needs k passes over the data for initialisation
 - In big data applications, k is typically large (e.g., 1000) \rightarrow not scalable!
- A promising solution
 - k -means++ samples **just** one point per iteration
 - What if we **oversample** by sampling each point independently with a larger probability?
 - Equivalent to updating the distribution less frequently
 - Coarser sampling \rightarrow k -means|| [Bahmani et al. '12]

k -means || Initialisation

$k=4$, oversampling factor $L=3$

Cluster the intermediate centres

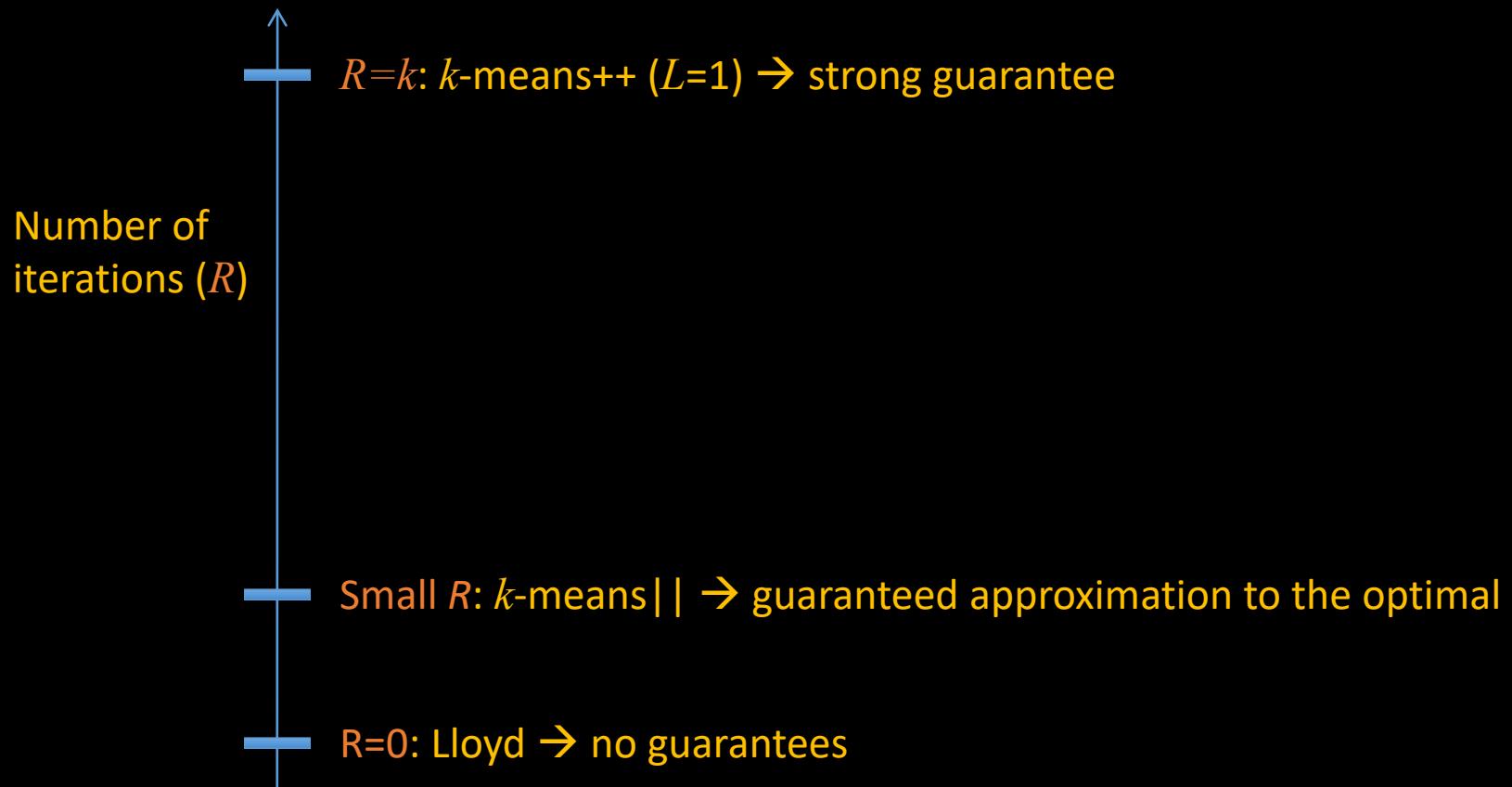


k -means|| [Bahmani et al. '12]

- Choose the oversampling factor $L > 1$
- Initialise C to an arbitrary set of points
- For R iterations do:
 - Sample each point x in X independently with probability $p_x = Ld(x, C)^2 / SSE_X(C)$.
 - Add all the sampled points to C
- Cluster the intermediate centres in C using k -means++
- Benefits over k -means++
 - Less susceptible to noisy outliers
 - More reduction in the number of Lloyd iterations

k -means||: Relationships

- An interpolation between Lloyd and k -means++



Week 8 Contents / Objectives

- Introduction to Cluster Analysis
- k -means Clustering
- Scalable k -means
- k -means in Spark & Limitations

k-means in Spark

- Implemented in the RDD API Mllib - Kmeans
- Scala ml code: [org/apache/spark/ml/clustering/KMeans.scala](https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/clustering/KMeans.scala)

```
29 import org.apache.spark.ml.util._  
30 import org.apache.spark.ml.util.Instrumentation.instrumented  
31 import org.apache.spark.mllib.clustering.{DistanceMeasure, KMeans => MLlibKMeans, KMeansModel => MLlibKMeansModel}  
32 import org.apache.spark.mllib.linalg.{Vector => OldVector, Vectors => OldVectors}  
33 import org.apache.spark.mllib.linalg.VectorImplicits._  
34  
35 . . . . .  
36 . . . . .  
37 . . . . .  
38 . . . . .  
39 . . . . .  
40 . . . . .  
41 . . . . .  
42 . . . . .
```

- Scala mllib: [org/apache/spark/ml/clustering/KMeans.scala](https://github.com/apache/spark/blob/master/mllib/src/main/scala/org/apache/spark/ml/clustering/KMeans.scala)

```
33 /**  
34  * K-means clustering with a k-means++ like initialization mode  
35  * (the k-means|| algorithm by Bahmani et al).  
36  *  
37  * This is an iterative algorithm that will make multiple passes over the data, so any RDDs given  
38  * to it should be cached by the user.  
39  */  
40 @Since("0.8.0")  
41 class KMeans private (  
42     private var k: Int,  
43     private var maxIterations: Int = 20,  
44     private var distanceMeasure: DistanceMeasure = EuclideanDistance,  
45     private var initialCentroids: Option[Vector] = None,  
46     private var seed: Long = 1L,  
47     private var instrumentation: Instrumentation = null,  
48     private var numPartitions: Int = 2,  
49     private var cacheDuration: Duration = Duration.Inf,  
50     private var cacheSize: Int = 1000000000,  
51     private var cacheType: CacheType = CacheType.LRU  
52 ) extends Estimator[KMeansModel] {  
53     private val logger = LoggerFactory.getLogger(KMeans.class)  
54  
55     def setK(k: Int): KMeans = {  
56         this.k = k  
57         this
```

k -means API

- **k**: the number of desired clusters
- **maxIter**: the maximum number of iterations
- **initMode**: random or via k-means|| (default) initialization
- **initSteps**: the number of steps in the k-means|| algorithm (default=2, oversampling factor $L = 2 \times k$)
- **tol**: the distance threshold for checking convergence
- **seed**: the random seed
- **distanceMeasure**: Euclidean (default) or cosine dist measure
- **weightCol**: optional weighting of data points

```
375     // On each step, sample  $2 * k$  points on average with probability proportional
376     // to their squared distance from the centers. Note that only distances between points
377     // and new centers are computed in each iteration.
378     var step = 0
379     val bcNewCentersList = ArrayBuffer[Broadcast[_]]()
380     while (step < initializationSteps) {
```

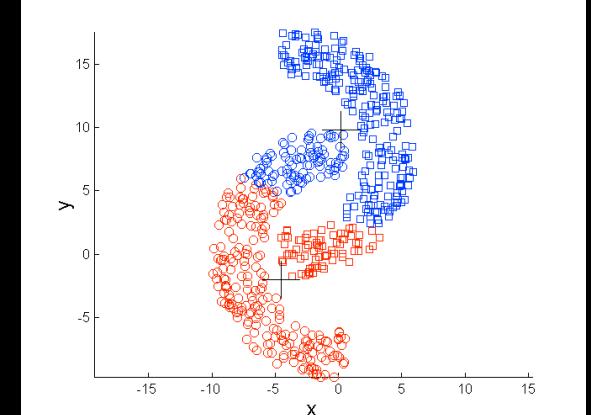
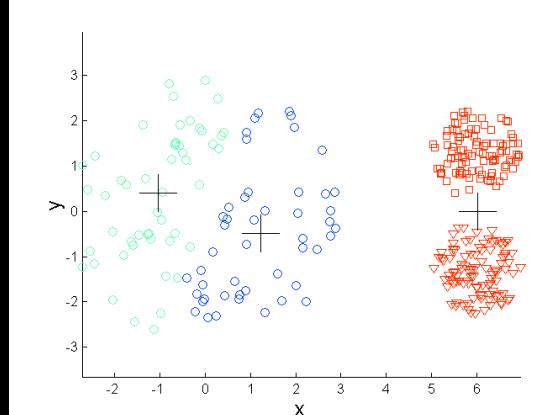
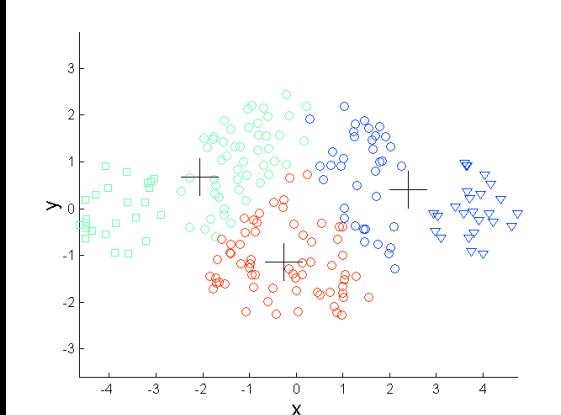
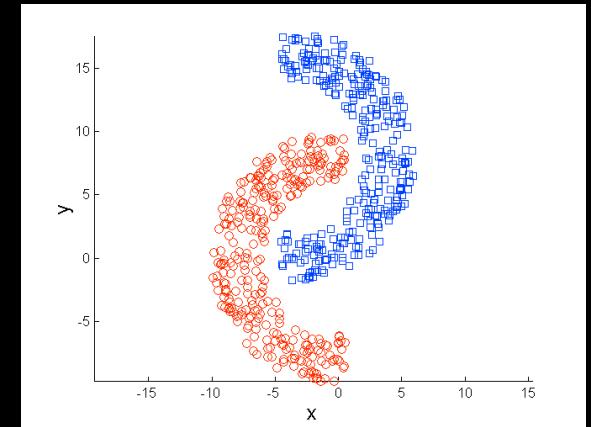
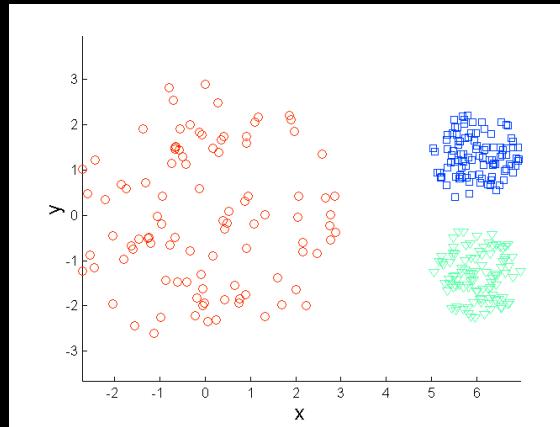
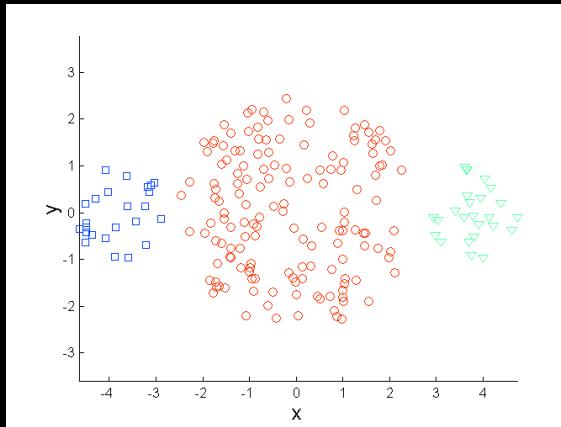
Remember to Cache

- Data need to be **cached** for high performance
- See from the source code

```
33  /**
34   * K-means clustering with a k-means++ like initialization mode
35   * (the k-means|| algorithm by Bahmani et al).
36   *
37   * This is an iterative algorithm that will make multiple passes over the data, so any RDDs given
38   * to it should be cached by the user.
39   */
40 @Since("0.8.0")
41 class KMeans private (
42
43     * Train a K-means model on the given set of points; `data` should be cached for high
44     * performance, because this is an iterative algorithm.
45     */
46
47     @Since("0.8.0")
48     def run(data: RDD[Vector]): KMeansModel = {
49         val instances = data.map(point => (point, 1.0))
50         val handlePersistence = data.getStorageLevel == StorageLevel.NONE
51
52         // ...
53
54         val centers = KMeansInternal.kMeans(
55             instances,
56             KMeansInternal.kMeansInitFunction,
57             KMeansInternal.kMeansUpdateFunction,
58             KMeansInternal.kMeansConvergenceFunction,
59             KMeansInternal.kMeansMaxIterations,
60             KMeansInternal.kMeansTolerance,
61             handlePersistence)
62
63         KMeansModel(centers, handlePersistence)
64     }
65 }
```

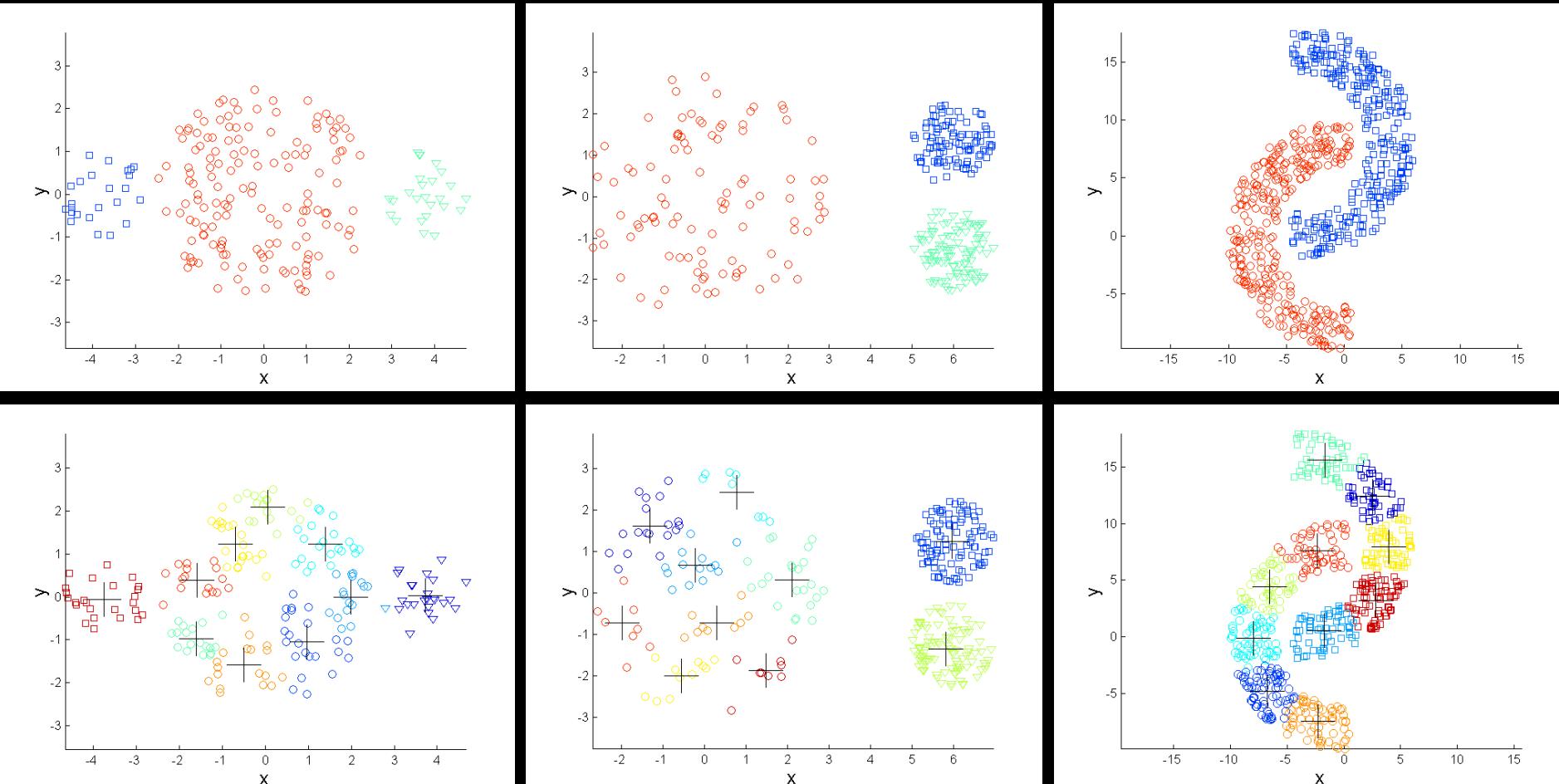
Limitations of k -means

- Problems when clusters are of differing sizes, densities, or non-globular shapes



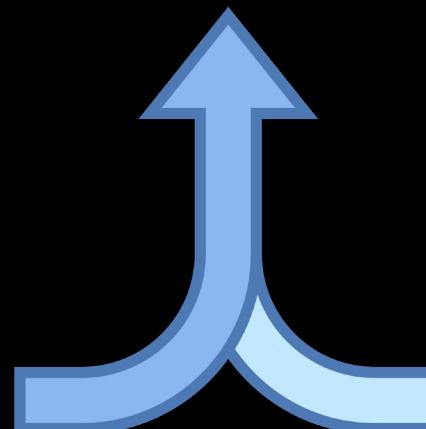
Overcoming k -means Limitations

- Use many clusters to find parts of clusters → put together



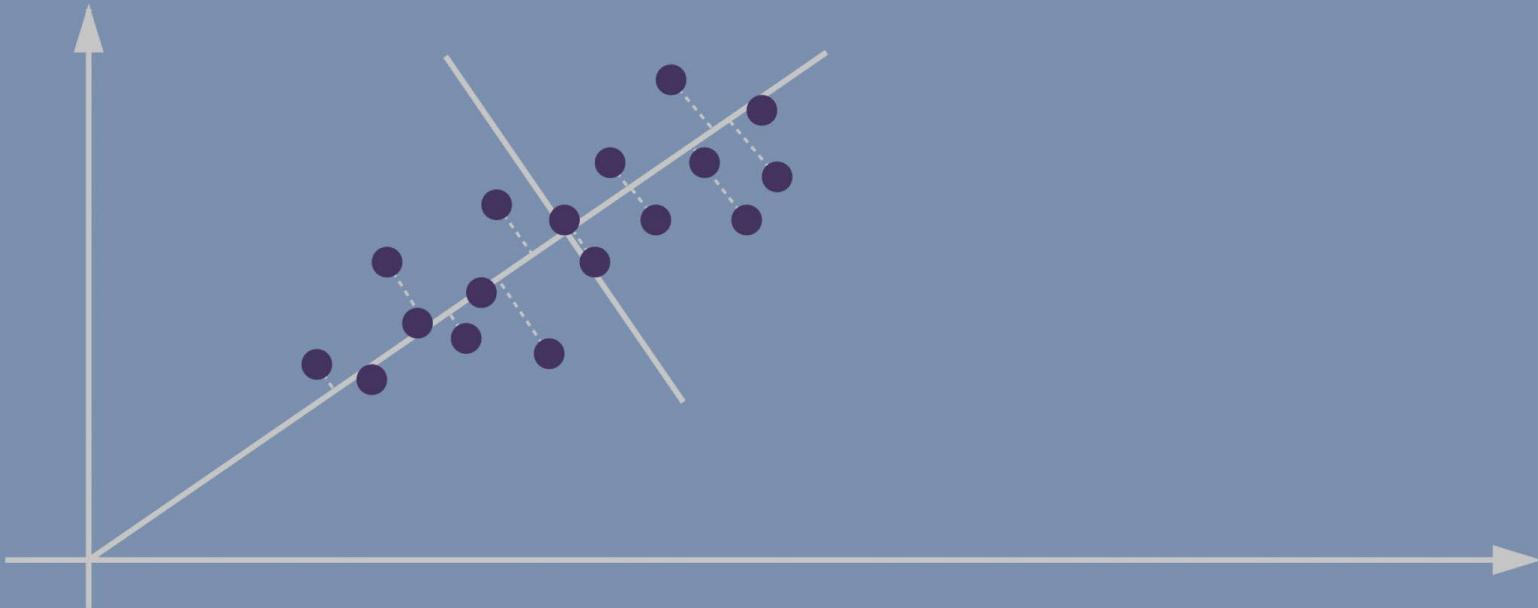
Pre-processing & Post-processing

- Pre-processing
 - Normalise the data (distance measure)
 - Eliminate outliers
- Post-processing
 - Eliminate small clusters that may represent outliers
 - Split **loose** clusters (with relatively high SSE)
 - Merge clusters that are **close** (with relatively low SSE)



Acknowledgement & References

- Acknowledgement
 - Some slides are adapted from 1) the [k-means|| slides](#) by Bahman Bahmani, Stanford University, 2012, and 2) Tan, Steinbach, Kumar's slides for the book [Introduction to Data Mining](#)
- References
 - [Chapter on clustering from the book above](#) (88 pages)
 - [k-means overview](#)
 - [k-means ++ paper](#)
 - [k-means || paper](#) and [video](#)



[PCA-06-scaled.jpg \(2560×1051\) \(perfectial.com\)](#)

Lecture 9: Scalable PCA for Dimensionality Reduction

[COM6012: Scalable ML](#) by [Haiping Lu](#)

YouTube Playlist: <https://www.youtube.com/c/HaipingLu/>

Week 9 Contents / Objectives

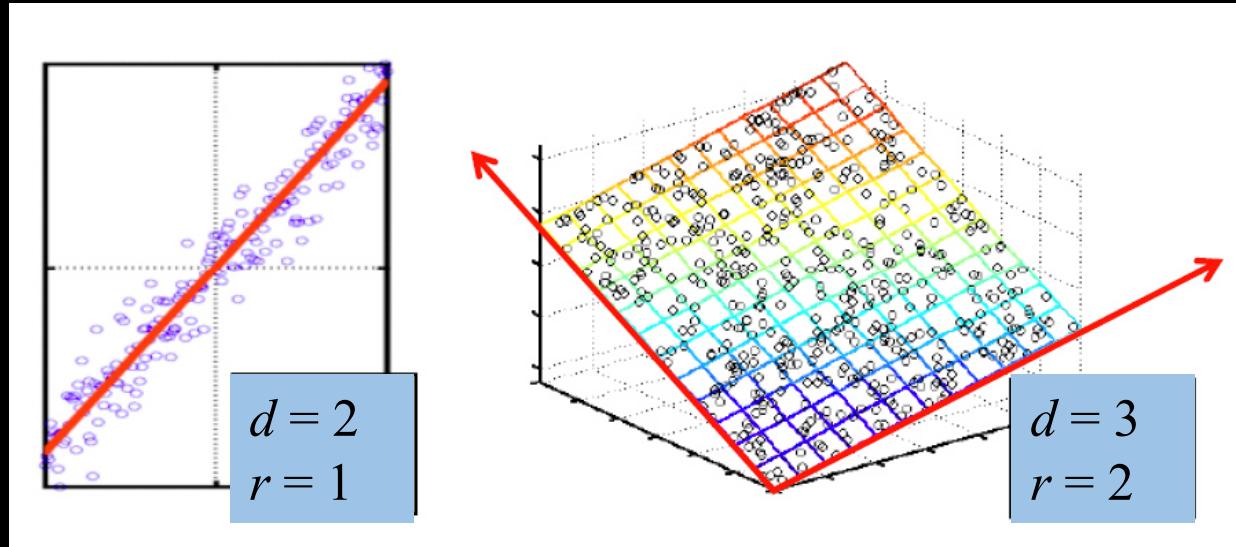
- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Week 9 Contents / Objectives

- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Motivation of Dimensionality Reduction

- Raw data: complex and high-dimensional
- Assumption: they lie on a low-dimensional subspace
 - Axes of this subspace → representation of the data
 - Simpler, more compact, showing interesting patterns



Utilities of Dimensionality Reduction

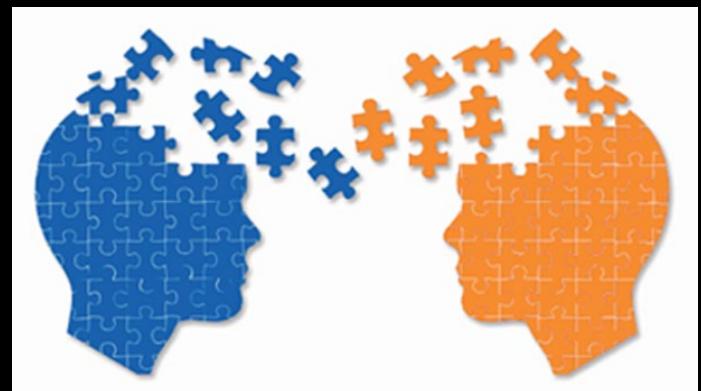
- Discover hidden correlations/topics
- Remove redundant/noisy features
- Interpretation and visualisation
- Easier storage and processing of the data



[owners-icebergs-blog-image-300x300.jpg \(resettorgrow.com\)](#)



[1*KvKlx9OnIxd0TfNxWKAY_g.jpeg \(480x320\) \(medium.com\)](#)



[Interpreting and Translation Blog: Image \(wordpress.com\)](#)

PCA → Variance Maximisation

- Input: n d -dimensional data points
- PCA algorithm
 - $X_0 \leftarrow n \times d$ data matrix, data point \rightarrow row vector x_i
 - X : subtract mean x from each row vector x_i in X_0
 - $\Sigma \leftarrow X^T X$: Gramian/scatter matrix for X
 - Find eigenvectors and eigenvalues of Σ
 - $U (d \times r) \leftarrow$ the top r eigenvectors (PCs)
- PCA features for y : $U^T y$ (dimension: $d \rightarrow r$)
 - Zero correlations, ordered by variance

Scalability Problem of PCA

- Input dimensionality → scatter matrix
 - Images: $100 \times 100 \rightarrow 10^4$; $1000 \times 1000 \rightarrow 10^6$
 - Scatter matrix Σ is of size d^2
 - $d = 10^4 \rightarrow \Sigma$ size 10^8
 - $d = 10^6 \rightarrow \Sigma$ size = 10^{12}
- Alternative: Singular Value Decomposition (SVD)
 - Efficient algorithms available
 - Often need just top r eigenvectors

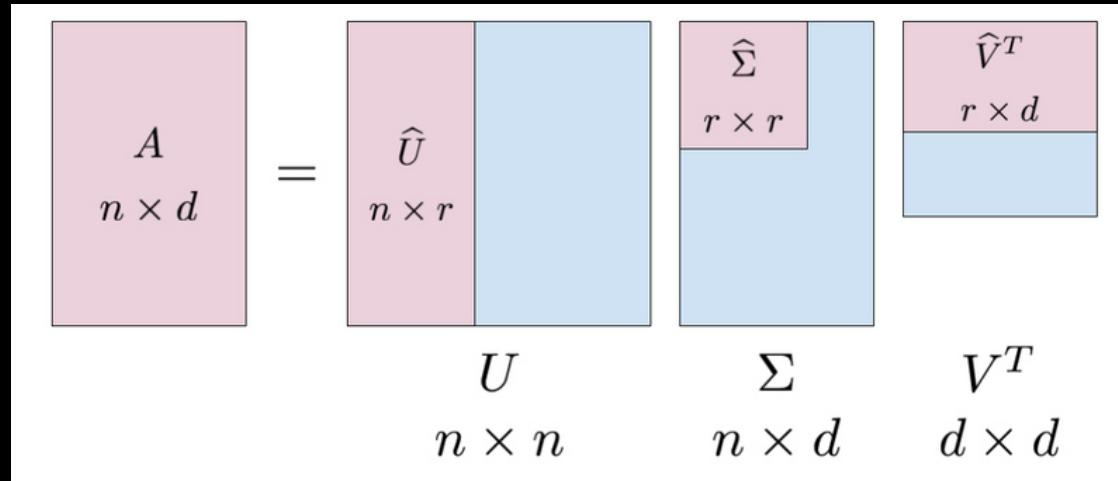
Week 9 Contents / Objectives

- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Singular Value Decomposition (SVD)

$$\mathbf{A}_{[n \times d]} = \mathbf{U}_{[n \times r]} \Sigma_{[r \times r]} (\mathbf{V}_{[d \times r]})^T$$

- \mathbf{A} : $n \times d$ matrix
- r : the rank of the matrix
- \mathbf{U} : $n \times r$ matrix, column orthonormal, $\mathbf{U}^T \mathbf{U} = \mathbf{I}$
- Σ : $r \times r$ diagonal matrix, strength of each factor
- \mathbf{V} : $d \times r$ matrix, column orthonormal, $\mathbf{V}^T \mathbf{V} = \mathbf{I}$



SVD \longleftrightarrow Eigen-decomposition

- SVD gives
 - $X = U \Sigma V^T$
- Eigen-decomposition gives
 - $B = X^T X = W \Lambda W^T$
- U, V, W : **orthonormal** $\rightarrow U^T U = I, V^T V = I, W^T W = I$
- Σ, Λ : diagonal
- Relationship:
 - $XX^T = U \Sigma V^T (U \Sigma V^T)^T = U \Sigma V^T (V \Sigma^T U^T) = U \Sigma^2 U^T$
 - $X^T X = V \Sigma^T U^T (U \Sigma V^T) = V \Sigma \Sigma^T V^T = V \Sigma^2 V^T$
 - $B = X^T X = W \Lambda W^T = V \Sigma^2 V^T$

PCA via SVD

- $X_0 \leftarrow n \times d$ data matrix, data point \rightarrow row vector x_i
- X : subtract mean x from each row vector x_i in X_0
- $U \Sigma V^T \leftarrow$ SVD of X
- The top r right singular vectors V of $X \rightarrow$ the PCs
- The singular values in $\Sigma =$ the square roots of the eigenvalues of $X^T X$

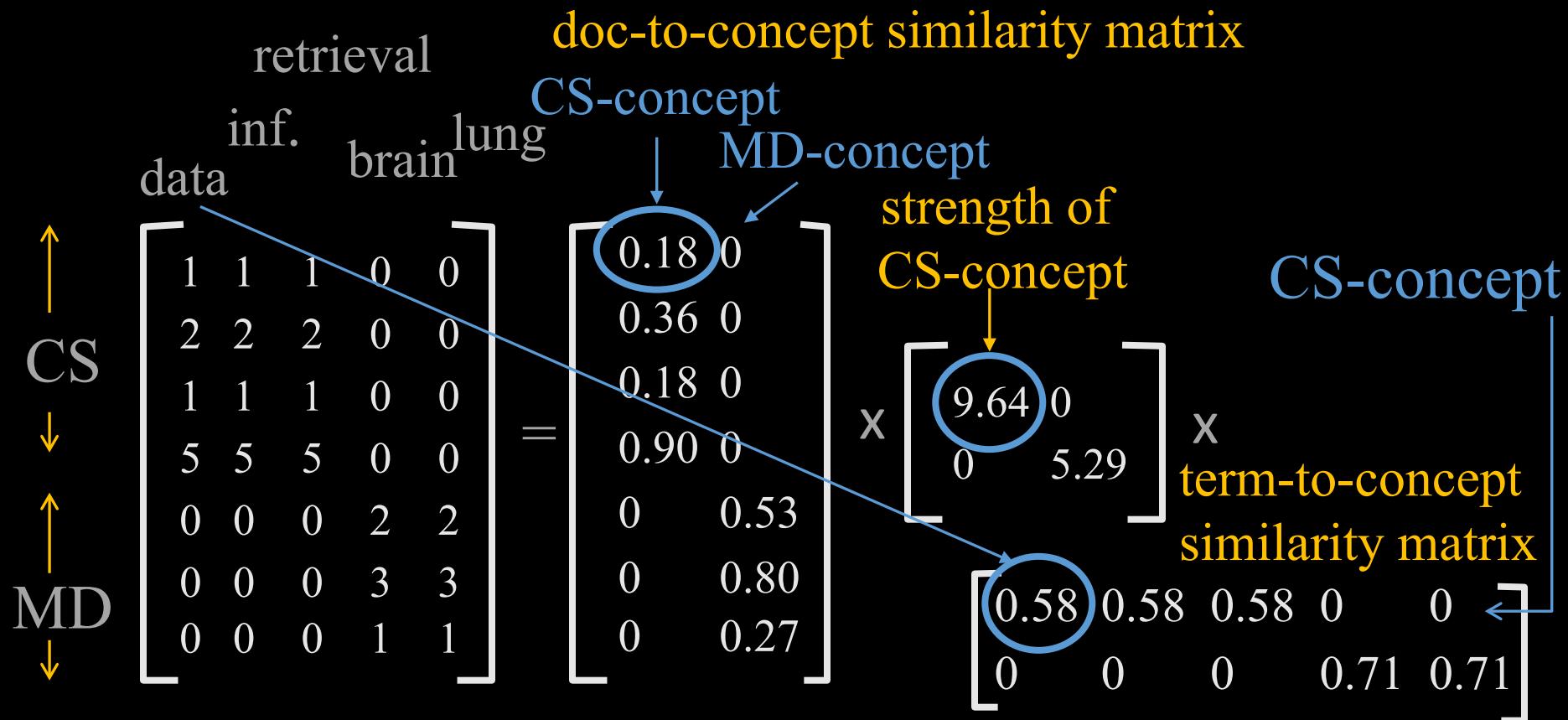
Example on a Document x Term

Term Document	data	information	retrieval	brain	lung
CS-TR1	1	1	1	0	0
CS-TR2	2	2	2	0	0
CS-TR3	1	1	1	0	0
CS-TR4	5	5	5	0	0
MED-TR1	0	0	0	2	2
MED-TR1	0	0	0	3	3
MED-TR1	0	0	0	1	1

- $d = 5$ but $r=2 \rightarrow$ two bases $[1 1 1 0 0]$ & $[0 0 0 1 1]$
- U: document-to-concept similarity matrix
- V: term-to-concept similarity matrix
- Σ : its diagonal elements \rightarrow strength of each concept

Interpretation

Term Document	data	information	retrieval	brain	lung
CS-TR1	1	1	1	0	0
CS-TR2	2	2	2	0	0
CS-TR3	1	1	1	0	0
CS-TR4	5	5	5	0	0
MED-TR1	0	0	0	2	2
MED-TR1	0	0	0	3	3
MED-TR1	0	0	0	1	1



SVD – Dimensionality Reduction

- To reduce the dimensionality further (3 zero singular values have already been removed)
 - Best rank-1 approximation \rightarrow

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 2 & 2 \\ 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 1 & 1 \end{bmatrix} = \begin{bmatrix} 0.18 & 0 \\ 0.36 & 0 \\ 0.18 & 0 \\ 0.90 & 0 \\ 0 & 0.53 \\ 0 & 0.80 \\ 0 & 0.27 \end{bmatrix} \times \begin{bmatrix} 9.64 & 0 \\ 0 & 5.29 \end{bmatrix} \times \begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 2 & 2 & 2 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The diagram illustrates the Singular Value Decomposition (SVD) of a 7x5 matrix. The matrix is shown as a product of three matrices: U (left singular vectors), S (singular values), and V (right singular vectors). The S matrix is highlighted with yellow lines and contains non-zero entries at positions (1,1), (2,1), (3,1), (4,1), (5,1), (6,2), and (7,3). The V matrix has a yellow 'X' over its second column. The U matrix has a yellow 'X' over its third row.

Week 9 Contents / Objectives

- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Three PCA APIs in Spark MLLib

- DataFrame-based API – [PCA \(source code, Scala doc\)](#)
 - `pyspark.ml.feature.PCA(k=None, inputCol=None, outputCol=None)`
- RDD-based API – [RowMatrix \(source code, Scala doc\)](#)
 - `computePrincipalComponents(k)`
 - **Scalable:** `computeSVD(k, computeU=False, rCond=1e-09)`

```
465 @Since("1.6.0")
466 def computePrincipalComponentsAndExplainedVariance(k: Int): (Matrix, Vector) = {
467   val n = numCols().toInt
468   require(k > 0 && k <= n, s"k = $k out of range (0, n = $n]")
469
470   if (n > 65535) {
471     val svd = computeSVD(k)
472     val s = svd.s.toArray.map(eigValue => eigValue * eigValue / (n - 1))
473     val eigenSum = s.sum
474     val explainedVariance = s.map(_ / eigenSum)
```

SVD in Spark MLlib (RDD)

- $U: m \times k$; $\Sigma : k \times k$; $V: n \times k$
- Assumption: n (dimensionality) < m (# samples)
- Different methods based on computational cost
 - If n is small ($n < 100$) or k is large compared with n ($k > n/2$), compute $A^T A$ first and then compute its top eigenvalues and eigenvectors **locally** on the driver
 - Otherwise, compute $(A^T A)v$ in a **distributive** way and send it to ARPACK to compute $(A^T A)$'s top eigenvalues/eigenvectors on the driver node

Selection of SVD Computation

```
334     if (n < 100 || (k > n / 2 && n <= 15000)) {
335         // If n is small or k is large compared with n, we better compute the Gramian matrix first
336         // and then compute its eigenvalues locally, instead of making multiple passes.
337         if (k < n / 3) {
338             SVDMode.LocalARPACK
339         } else {
340             SVDMode.LocalLAPACK
341         }
342     } else {
343         // If k is small compared with n, we use ARPACK with distributed multiplication.
344         SVDMode.DistARPACK
345     }
346     case "local-svd" => SVDMode.LocalLAPACK
347     case "local-eigs" => SVDMode.LocalARPACK
348     case "dist-eigs" => SVDMode.DistARPACK
349     case _ => throw new IllegalArgumentException(s"Do not support mode $mode.")
```

Week 9 Contents / Objectives

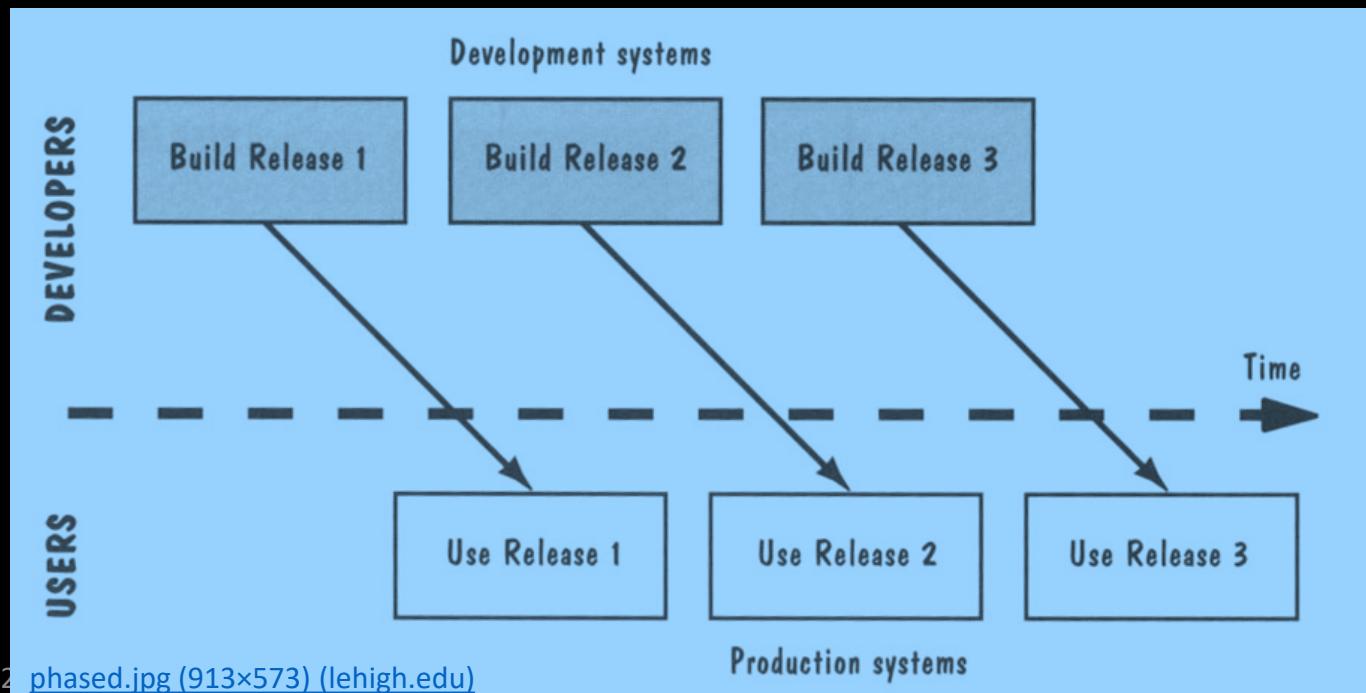
- Scalability Problem of PCA
- PCA via SVD
- Scalable PCA in Spark
- Software Development Life Cycle

Software and Changes

- A virtue of software: relatively easy to change
 - Otherwise, it might as well be hardware
- Planning for change
 - Good *comments* describe meaning of code to facilitate and reduce the cost of software maintenance
 - *Modularity* help manage change because modules help to isolate and localize change

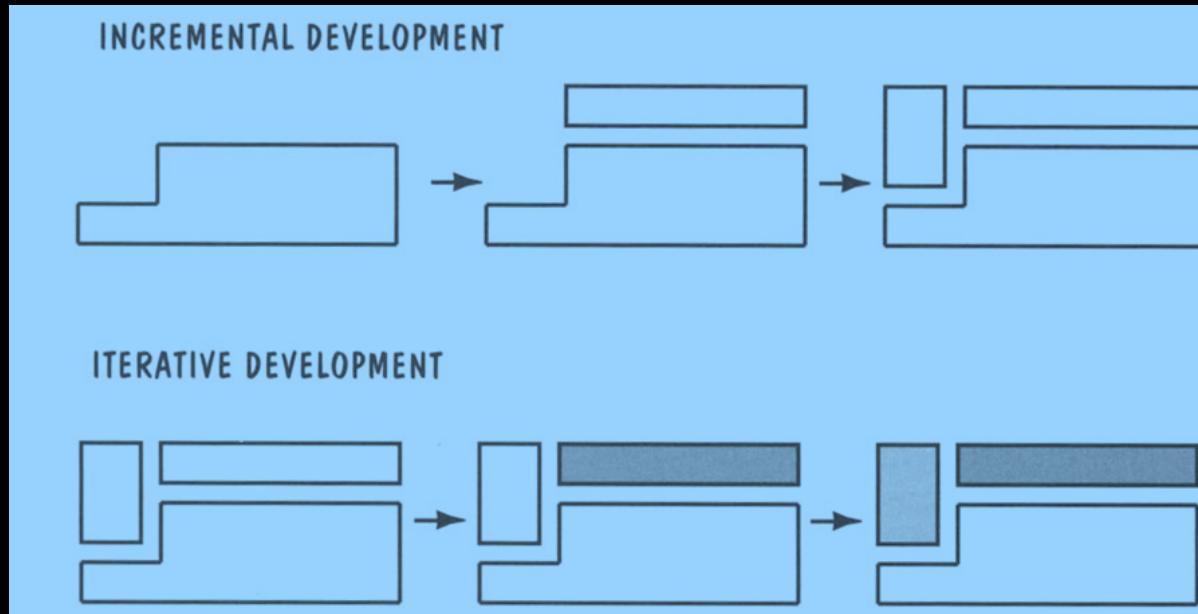
Phased Development

- Reduce cycle time, deliver in pieces, let users have some functionality while developing the rest
- Two or more systems in parallel
 - The **operational/production** system in use by customers
 - The **development** system to replace the current release



Iterative/Incremental Development

- **Incremental:** partition a system by functionality
 - Early release: small, functional subsystem
 - Later releases: add functionality
- **Iterative:** improve overall system in each release

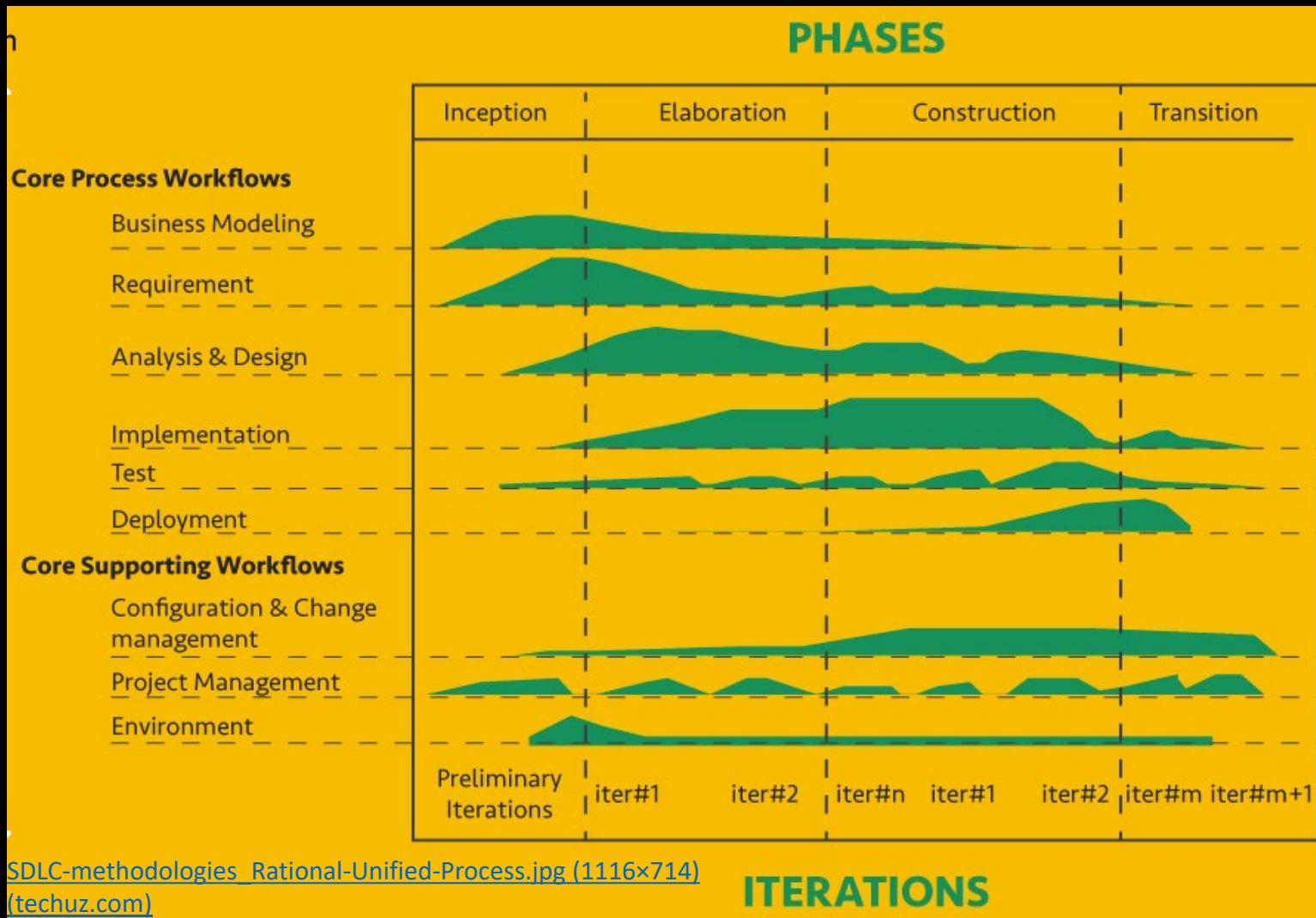


[iterative.jpg \(902x539\) \(lehigh.edu\)](#)

Lifecycle Phases

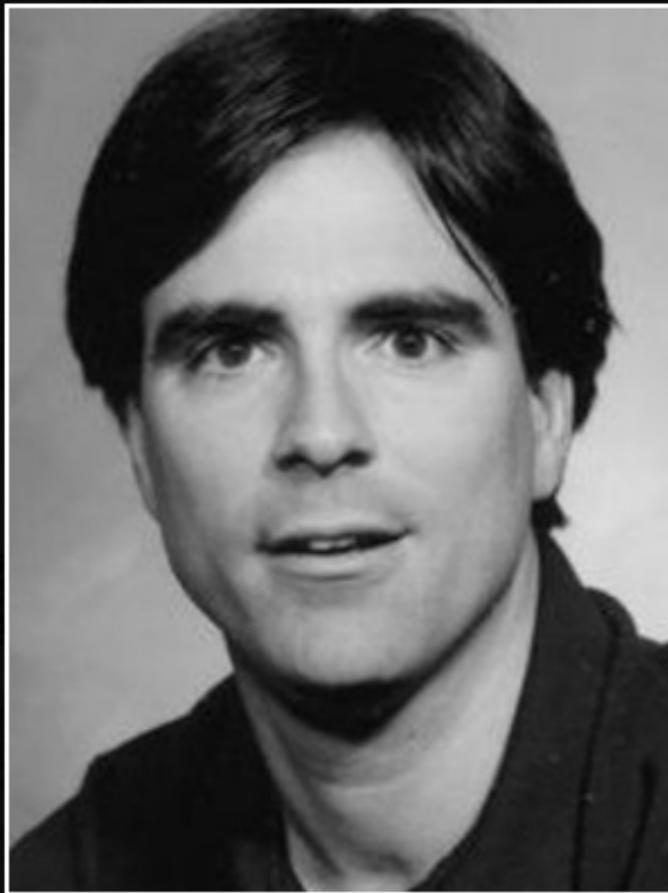
- **Inception:** rationale, scope, and vision
- **Elaboration:** “Design/Details”
 - detailed requirements and high-level analysis/design
- **Construction** – “Do it”
 - build software in increments, tested and integrated, each satisfying a subset of the requirements
- **Transition** – “Deploy it”
 - beta testing, performance tuning, and user training
- **Phases:** NOT the classical requirements/design/coding/implementation processes
- **Phases iterate over many cycles**

Phases: Iterative & Incremental



Work on Your Project

- Get a **small subset** or a reduced version to study, develop, debug, and test
- Break down big/difficult problem into smaller/easier sub-steps (**avoid black-box debugging**)
- Be structured, organised, and logical
- Keep good documentation
- Get help online (e.g. search) and keep the references



Engineering isn't about perfect solutions; it's about doing the best you can with limited resources.

— *Randy Pausch* —

AZ QUOTES

[Randy Pausch quote: Engineering isn't about perfect solutions; it's about doing the best... \(azquotes.com\)](https://www.azquotes.com/quote/Engineering_isn_t_about_perfect_solutions_it_s_about_doing_the_best..._Randy_Pausch)

Acknowledgement & References

- Acknowledgement
 - Some slides are adapted from the [MMDS book](#) slides and the slides on software process life cycles by [Glenn Blank](#)
- References
 - [Chapter 11](#) of the [MMDS book](#)