Lecture 1
Fundamentals of Big Data and Data Analytics

Dr. Lydia Wahid

Agenda

- What is Big Data?
- Big Data characteristics
- Data types processed by Big Data solutions
- Basic Concepts
- What is Data Analytics?
- Categories of Data Analytics
- Adoption of Analytics in Business
- Data Analytics Lifecycle
- Applications

What is Big Data?

What is Big Data?

- Big Data is a field dedicated to the **processing**, **analysis**, and **storage** of large collections of data.
- By large collections of data, we mean, collections of datasets whose volume, velocity or variety is so large that it is difficult (or even impossible) to process, analyze, and store using the traditional techniques.

What is Big Data?

- The amount of data worldwide has been growing ever since the invention of the World Wide Web.
- We have **search engines** that need to look through billions of websites to return a particular information.
- Then came the **social networks** that have billions of users that create all types of transactions and content.

What is Big Data?

- In addition, **businesses** and **governmental institutions** record every transaction of each customer, vendor, and supplier and thus have been accumulating data.
- We have also the data generated by **sensors** embedded in devices such as smartphones, energy smart meters, automobiles.
- Also, we have the data generated daily from **satellite imagery** and **communication networks**.

What is Big Data?

- The result is an explosive growth in the amount of data.
- This phenomenal growth of data generation means that the amount of data in a single repository can be numbered in terabytes (1,024 gigabytes) or petabytes (1,024 terabytes).
- The term *big data* also refers to such massive amounts of data.

Big Data characteristics

Big Data characteristics

- The five Big Data characteristics (named 5 V's) can be used to help differentiate data categorized as "Big" from other forms of data:
  1. Volume
  2. Velocity
  3. Variety
  4. Veracity
  5. Value

Big Data characteristics

1. **Volume:**
   - The volume of data refers to the size of data managed by the system.
   - In Big Data environments, high data volumes impose different data storage and processing demands, as well as additional data preparation, organization and management processes.
2. **Velocity:**
   - The velocity of data refers to speed at which data is created, accumulated, ingested, and processed.
   - In Big Data environments, data can arrive at fast speeds, and enormous datasets can accumulate within very short periods of time.

Big Data characteristics

3. **Variety:**
   - Data variety refers to the multiple formats and types of data that need to be supported by Big Data solutions.
   - Data variety brings challenges for enterprises in terms of data integration, transformation, processing, and storage.
4. **Veracity:**
   - Veracity refers to the quality or fidelity of data.
   - Data that enters Big Data environments needs to be assessed for quality, which can lead to data processing activities to resolve invalid data and remove noise.

Big Data characteristics

5. **Value:**
   - Value is defined as the usefulness of data for an enterprise.
   - The value characteristic is related to the **veracity** characteristic in that the higher the data fidelity, the more value it holds for the business.
   - Value is also dependent on **how long data processing takes**; the longer it takes for data to be turned into meaningful information, the less value it has for a business.

Big Data characteristics

- The following figure provides two illustrations of how value is impacted by the veracity of data and the time of generated results:

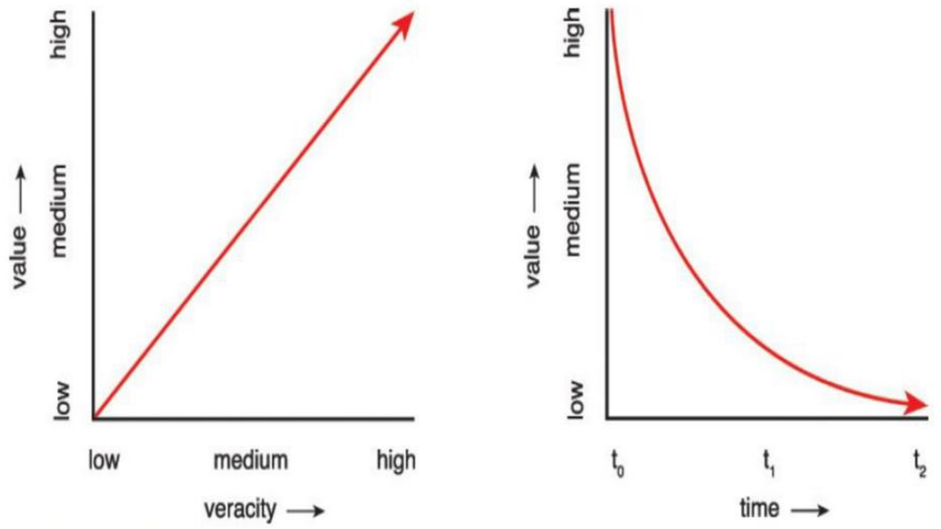


Figure 1.15 Data that has high veracity and can be analyzed quickly has more value to a business.

Data types processed by Big Data solutions

Data types processed by Big Data solutions

- These data types refer to the internal organization of data and are sometimes called data formats. The primary types of data are:
  1. Structured data
  2. Unstructured data
  3. Semi-structured data

Data types processed by Big Data solutions

1. Structured data:

- Structured data is data that adheres to a pre-defined data model or schema and is often stored in tabular form.
- It is used to capture relationships between different entities and is therefore most often stored in a relational database.

	branch	employee_type	number
1	Wichita Falls	Back Office	19
2	Wichita Falls	Credit Specialist	20
3	Wichita Falls	Financial Sevices Sales	16
4	Wichita Falls	Business Development Manager	1
5	Wichita Falls	Head of Sales Group	2
6	San Antonio	DSA	1
7	San Antonio	Back Office	56
8	San Antonio	Deputy Regional Director	2
9	San Antonio	Credit Specialist	96
10	San Antonio	Financial Sevices Sales	20

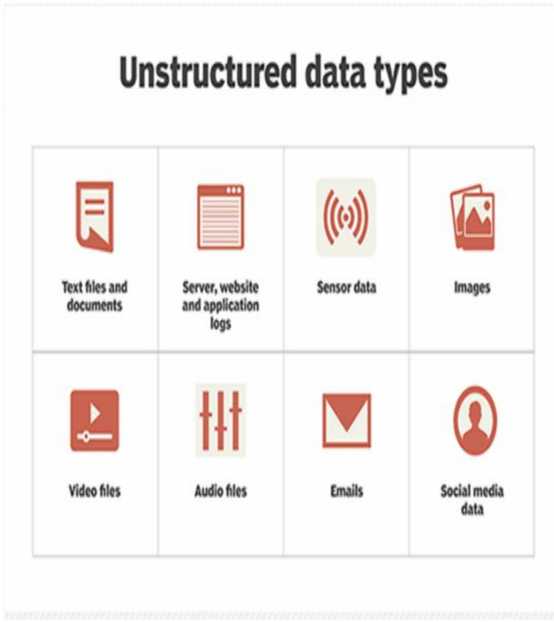
Data types processed by Big Data solutions

2. Unstructured data:

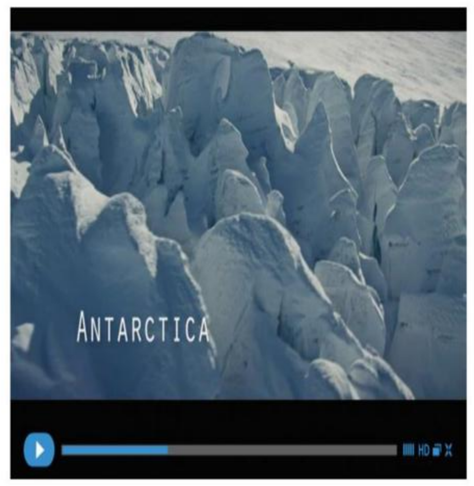
- Data that does not conform to a data model or data schema is known as unstructured data.
- One of the most common types of unstructured data is text collected in a wide range of forms, including Word documents, email messages, PowerPoint presentations, survey responses, transcripts of call center interactions, and posts from blogs and social media sites.
- Other types of unstructured data include images, audio and video files

Data types processed by Big Data solutions

2. Unstructured data:



Example of unstructured data:
video about Antarctica expedition



Data types processed by Big Data solutions

3. **Semi-structured data:**
   - Semi-structured data has a defined level of structure and consistency, but is not relational in nature.
   - XML and JSON files are common forms of semi-structured data.
   - Due to the textual nature of this data and its conformance to some level of structure, it is more easily processed than unstructured data.

```
        rdf:resource="#_55D3DE366B2AD032" />
    </cim:Substation>
  - <cim:EnergyConsumer rdf:ID="_2963867E4A4B1669">
      <cim:EnergyConsumer.pfixed>19.78</cim:EnergyConsumer.pfixed>
      <cim:EnergyConsumer.qfixed>06.10</cim:EnergyConsumer.qfixed>
      <cim:EnergyConsumer.LoadArea
        rdf:resource="#_9C16D2456B178B75" />
      <cim:Equipment.MemberOf_EquipmentContainer
        rdf:resource="#_A9D1427B3784CD78" />
      <cim:Naming.name>4711</cim:Naming.name>
```


Basic Concepts

Basic Concepts

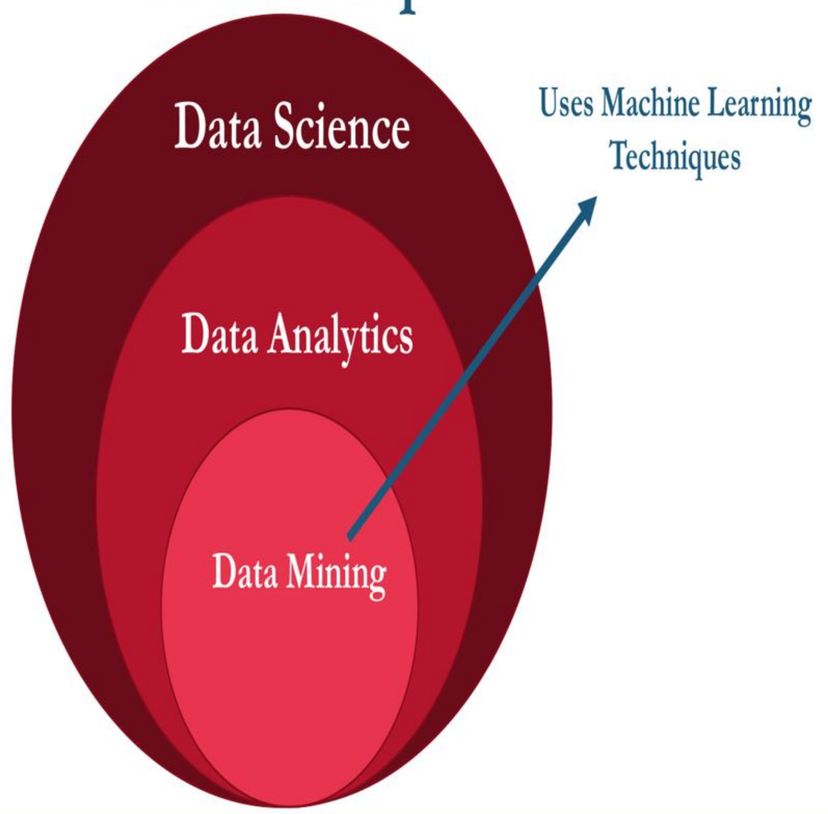
- The use of the terms "**Data Science**", "**Data Analytics**", and "**Data Mining**" are becoming increasingly common along with "**Big Data**."

- **Data Science:**
  - Data Science is a field focused around all what is related to data.
  - The term "Science" implies knowledge gained by systematic study.

Basic Concepts

- **Data Analytics:** Take Actions
  - Data analytics is defined as the application of computer systems to analyze large data sets for the **support of decisions** and **taking the right actions**.
  - Data analytics helps analysts **draw conclusions** from the data.
- **Data Mining:** Extract Knowledge
  - The goal of data mining is to **extract knowledge** from data.
  - In this context, knowledge is defined as **interesting patterns** that are generally valid, novel, useful, and understandable to humans.
  - Data Analytics uses Data mining techniques to help achieve its goals.

Basic Concepts



Basic Concepts

- Therefore, the sequence of progression looks as follows:

Basic Concepts

- **Example:**
  - A data engineer decides to look into a supermarket's raw sales data (**Big Data**).
  - After reviewing the data, the engineer discovers a high correlation of people buying burgers and fries on Sunday afternoon. (**Data Mining**)
  - Data analytics can look into the correlation of people buying burgers and fries on Sunday afternoon and offer valuable insights to create targeted advertising campaigns. (**Data Analytics**)

What is Data Analytics?

What is Data Analytics?

- Raw data does not have a meaning until it is processed into useful information.
- This information obtained is then organized and structured to infer knowledge about the system and/or its users, its environment, and its operations and progress towards its objectives, thus making the systems smarter and more efficient.
- Data Analytics is this process of creating information and knowledge from raw data to find actionable insights.

What is Data Analytics?

- **Data Analytics** is a broad term that includes the processes, technologies, frameworks and algorithms to extract meaningful insights from data.
- **Data Analytics** encompasses the management of the complete data lifecycle, which includes collecting, cleansing, organizing, storing, analyzing and governing data.

What is Data Analytics?

- **Data Analytics** enable data-driven *decision-making* with scientific backing so that decisions can be based on factual data and not simply on past experience or intuition alone.
- In **Big Data** environments, data analytics has developed methods that allow analytics to occur through the use of highly scalable distributed technologies and frameworks that are capable of analyzing large volumes of data from different sources.

Categories of Data Analytics

Categories of Data Analytics

- There are four general categories of analytics that are distinguished by the results they produce:
  - Descriptive Analytics
  - Diagnostic Analytics
  - Predictive Analytics
  - Prescriptive Analytics

Categories of Data Analytics

- **Descriptive Analytics:** *"What has happened?"*
  - Descriptive analytics are carried out to **answer questions about events that have already occurred.**
  - These help in describing patterns in the data and present the data in a summarized form.
  - Sample questions can include:
    - What was the sales volume over the past 12 months?
    - What is the number of support calls received as categorized by severity and geographic location?
    - What is the monthly commission earned by each sales agent?

Categories of Data Analytics

- **Diagnostic Analytics:** *"Why did it happen?"*
  - Diagnostic analytics aim to **determine the cause of a phenomenon** that occurred in the past using questions that focus on the reason behind the event.
  - The goal of this type of analytics is to **determine what information is related to the phenomenon** in order to enable answering questions that seek to determine why something has occurred.
  - Sample questions can include:
    - Why were Q2 sales less than Q1 sales?
    - Why have there been more support calls originating from the Eastern region than from the Western region?
    - Why was there an increase in patient re-admission rates over the past three months?

Categories of Data Analytics

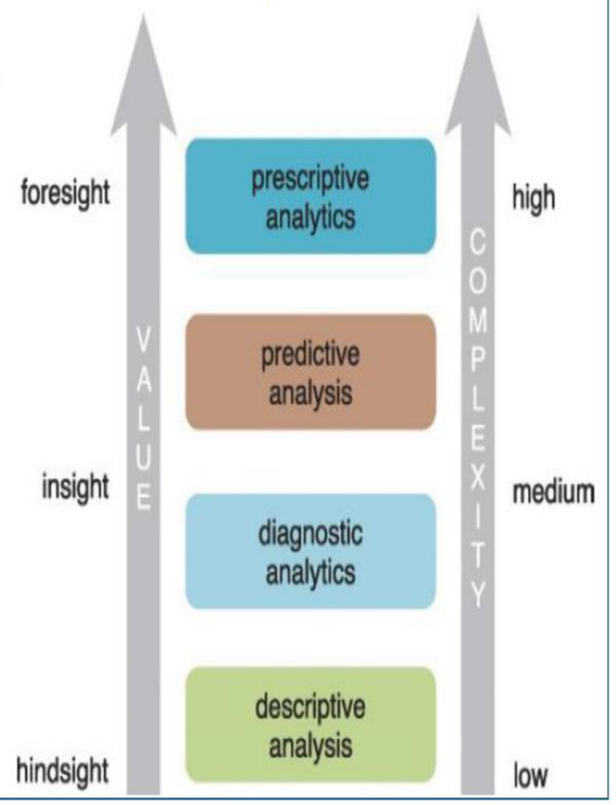
- **Predictive Analytics:** *"What is likely to happen?"*
  - Predictive analytics includes **predicting the occurrence of an event** or the likely outcome of an event or **forecasting the future values** using prediction models.
  - Sample questions can include:
    - What are the chances that a customer will default on a loan if they have missed a monthly payment?
    - What will be the patient survival rate if medicine B is administered instead of medicine A?
    - If a customer has purchased Products A and B, what are the chances that they will also purchase Product C?

Categories of Data Analytics

- **Prescriptive Analytics:** *"What can we do to make it happen??"*
  - Prescriptive analytics build upon the results of predictive analytics by **prescribing actions that should be taken.**
  - Prescriptive analytics uses multiple prediction models to **predict various outcomes and the best course of action for each outcome.**
  - Sample questions can include:
    - Among three medicines, which one to produce?
    - When is the best time to trade a particular stock?

Categories of Data Analytics

- The shown figure demonstrates the increase in **Value** and **Complexity** from descriptive to prescriptive analytics.
- **Hindsight:** understanding of a situation after it has happened.
- **Insight:** gain an accurate and deep understanding of something.
- **Foresight:** the ability to predict what will happen or what will be needed in the future.



Adoption of Analytics in Business

Terminologies

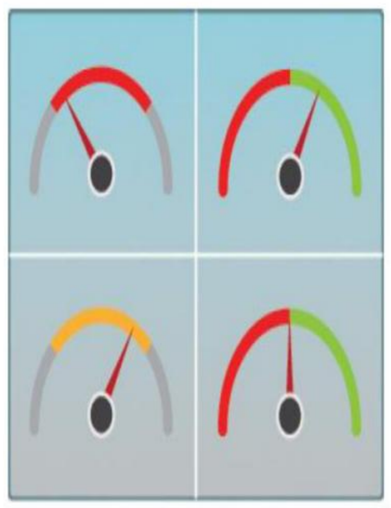
- **Business Intelligence (BI):**
  - BI enables an organization to **gain insight into the performance** of an enterprise **by analyzing data** generated by its business processes and information systems.
  - The results of the analysis can be used by management to steer the business in an effort to **correct detected issues** or otherwise **enhance organizational performance.**

Terminologies

- **Key Performance Indicators (KPIs):**
  - A KPI is a metric that can be used to **measure success** within a particular business context.
  - KPIs therefore act as **quantifiable reference** points for measuring a specific aspect of a business' overall performance.

Terminologies

- **Key Performance Indicators (KPIs):**
  - KPIs are often displayed via a **KPI dashboard** as shown in the figure.
  - The dashboard consolidates the display of multiple KPIs and compares the **actual measurements** with **threshold values** that define the acceptable value range of the KPI.



KPI dashboard

Terminologies

- **Key Performance Indicators (KPIs):**
  - Each department within an enterprise will use different KPI types to measure success based on specific business goals and targets.
  - Examples of KPIs:
    - Monthly Sales Growth,
    - Average Profit Margin,
    - Lifetime value of a customer,
    - Customer retention

Terminologies

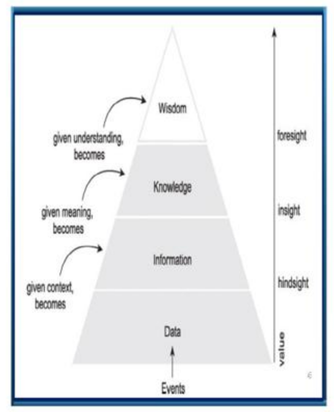
- **Performance Indicators (PIs):**
  - PIs are different in that they simply track the status of a specific business process while KPIs track whether you hit business objectives/targets, and metrics track processes.

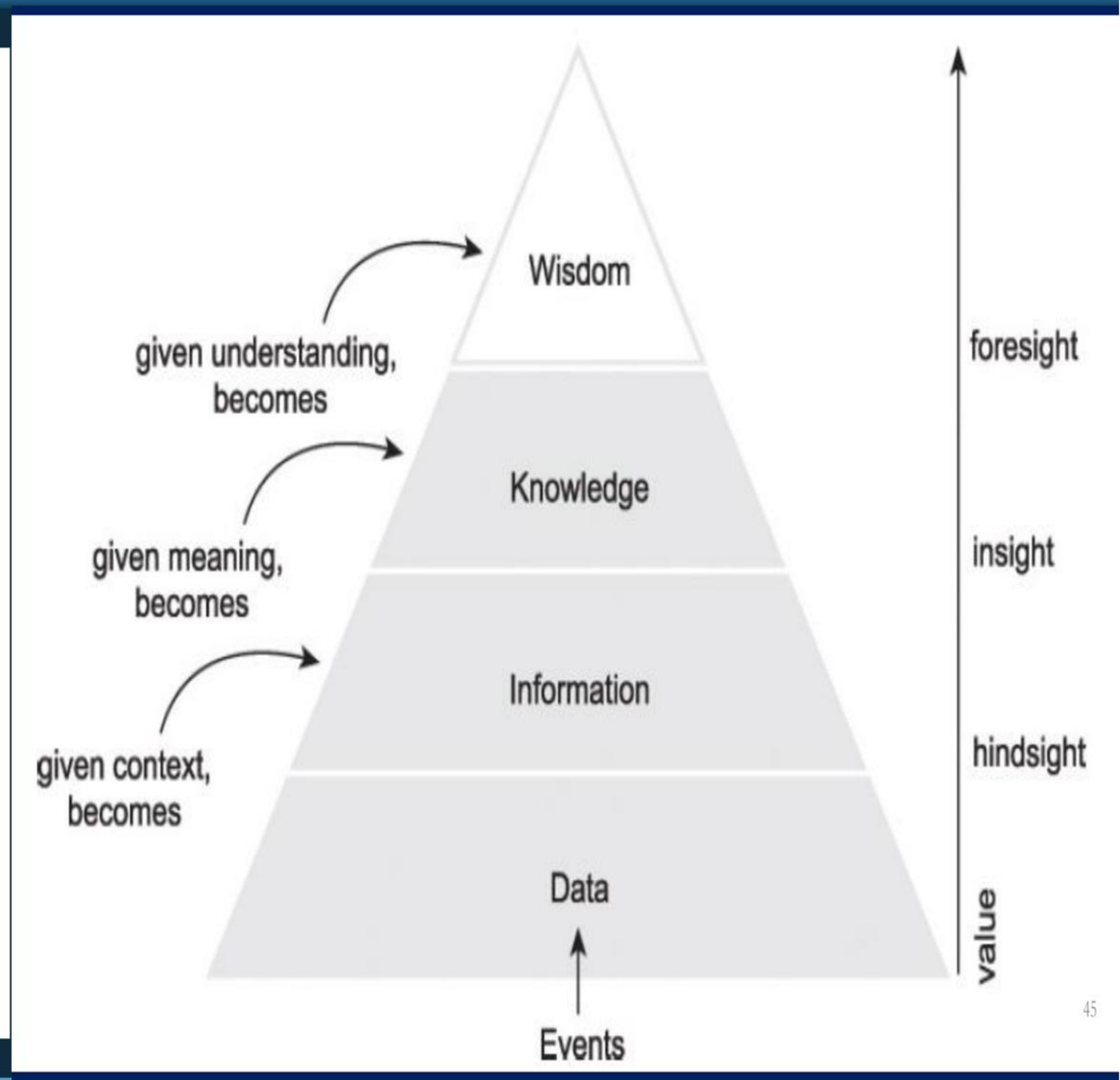
Terminologies

- **Critical Success factors (CSFs):**
  - Examples of Critical Success factors
    - Training and education.
    - Quality data and reporting.
    - Management commitment, customer satisfaction.
    - Staff Orientation.
    - Continuous improvement.

Adoption of Analytics in Business

- The transition from hindsight to foresight can be understood through the lens of the DIKW (**D**ata **I**nformation **K**nowledge **W**isdom) pyramid depicted in the figure:





Adoption of Analytics in Business

- **Examples of DIKW pyramid:**
  - Example1:
    - **Data:** 1815 feet, CN Tower, Toronto
    - **Information:** The CN Tower in Toronto is 1815 feet tall
    - **Knowledge:** Elevator can be used to ascend the building since CN tower is around 147 floors from the info of its height
    - **Wisdom:** If the elevator is not working, then do not use the stairs and go another day

Adoption of Analytics in Business

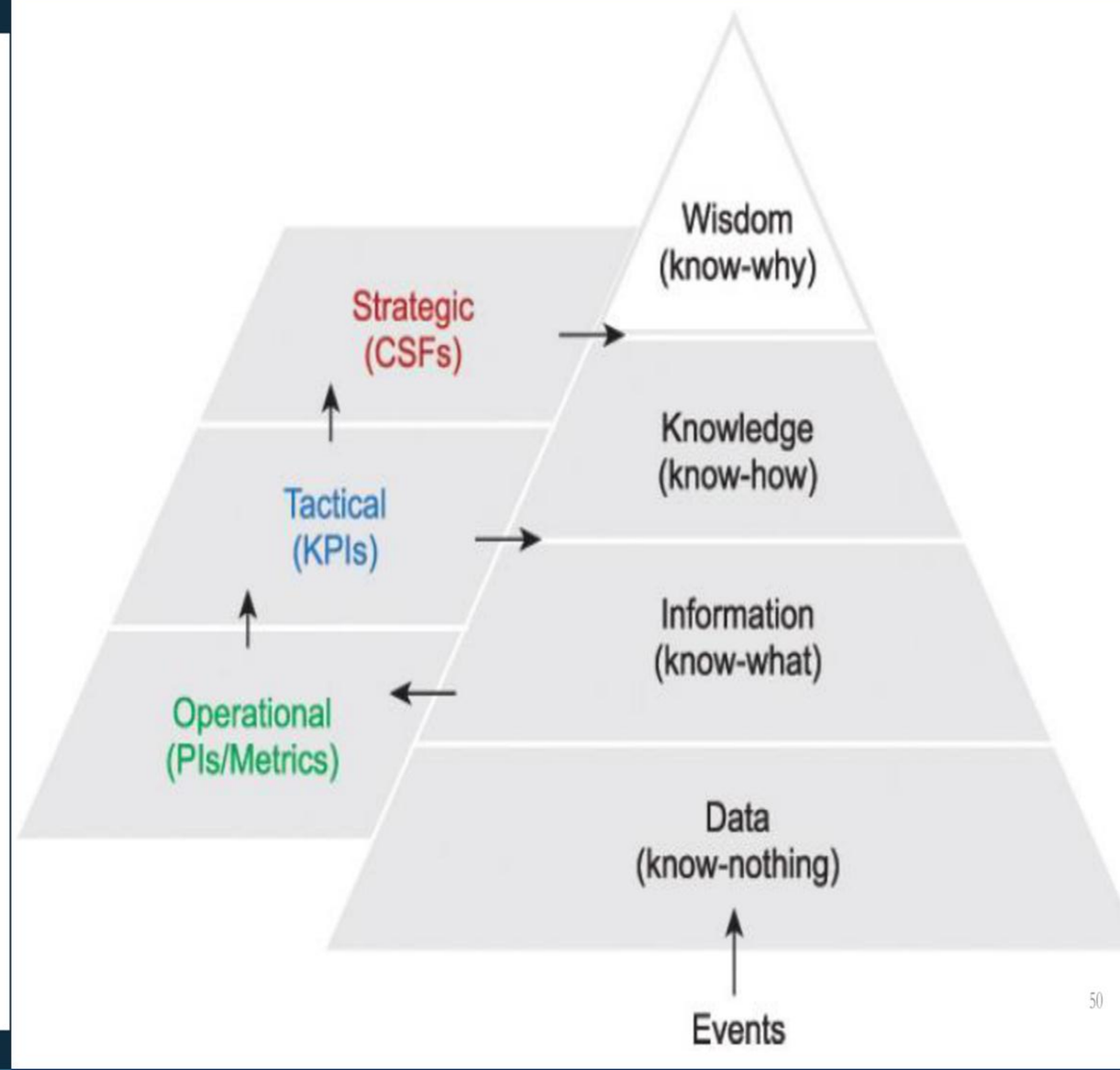
- **Examples of DIKW pyramid:**
  - Example2:
    - **Data:**
      - Employees – Ben; Anna; Mark; Kathy; Rose; Jack; Jane …
      - Departments – Accounting, Sales, Human Resources,….
    - **Information:**
      - Ben, Anna, Mark works in the Accounting Dept.
      - Kathy, Rose, Jack works in the Sales Dept.
      - Jane works in the Human Resources Dept.
    - **Knowledge:**
      - The company should hire a number of employees in all Depts.
    - **Wisdom:**
      - HR Dept. has only one Employee to handle the recruitment process, so we need to focus now on hiring for the HR Dept.

Adoption of Analytics in Business

- A business operates as a layered system—the top layer is the **strategic layer** occupied by C-level executives and advisory groups.
- The middle layer is the **tactical or managerial layer** that seeks to steer the organization in alignment with the strategy.
- The bottom layer is the **operations layer** where a business executes its core processes and delivers value to its customers.
- These three layers often exhibit a degree of independence from one another, but each layer's goals and objectives are influenced by another.

Adoption of Analytics in Business

- The following DIKW pyramid illustrates alignment with Strategic, Tactical and Operational corporate levels:



Adoption of Analytics in Business

- For instance, **at the operational level**, metrics are generated that simply report on ***what*** is happening in the business. *Data → Information.*
- **At the managerial level**, this information can be examined through the lens of corporate performance to answer questions regarding ***how*** the business is performing. *Information → Knowledge.*
- This information may be further enriched to answer questions regarding ***why*** the business is performing at the level it is. The **strategic layer** can then provide further insight to help answer questions of which strategy needs to change or be adopted in order to correct or enhance the performance. *Knowledge → Wisdom.*

Adoption of Analytics in Business

- Big Data has ties to business architecture at each of the organizational layers.
- Big Data enhances value as it helps convert data into information and provide meaning to generate knowledge from information.

Data Analytics Lifecycle

Data Analytics Lifecycle

- The Data analytics lifecycle can be divided into the following nine stages:
  1. Business Case Evaluation
  2. Data Identification
  3. Data Acquisition & Filtering
  4. Data Extraction
  5. Data Validation & Cleansing
  6. Data Aggregation & Representation
  7. Data Analytics
  8. Data Visualization
  9. Utilization of Analytics Results

Data Analytics Lifecycle

1. **Business Case Evaluation:**
   - Each Data analytics lifecycle must begin with a **well-defined business case** that presents a clear understanding of the *justification*, *motivation* and *goals* of carrying out the analysis.

2. **Data Identification:**
   - Identifying the datasets required for the analysis project and their sources.

Data Analytics Lifecycle

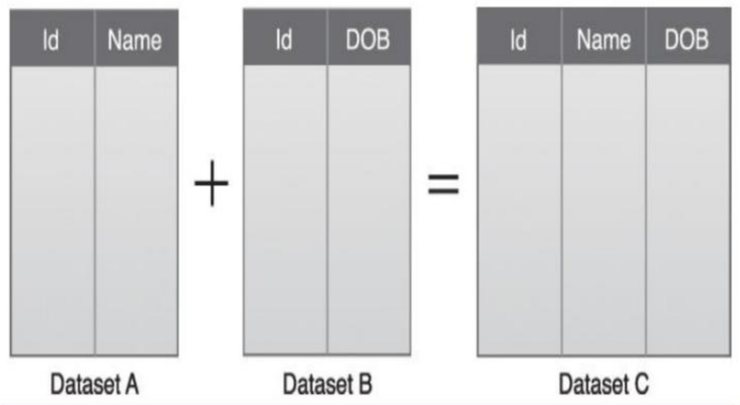
3. **Data Acquisition and Filtering:**
   - The data is **gathered** from all of the data sources that were identified during the previous stage.
   - The acquired data is then subjected to automated **filtering** for the removal of *corrupt data* or data that has been deemed to *have no value* to the analysis objectives.

Data Analytics Lifecycle

4. **Data Extraction:**
   - Some of the data may arrive in a format **incompatible** with the Big Data solution.
   - This stage is dedicated to extracting disparate data and transforming it into a format that the underlying Big Data solution can use for the purpose of the data analysis.
5. **Data Validation and Cleansing:**
   - The Data Validation and Cleansing stage is dedicated to **establishing often complex validation rules** and **removing any known invalid data.**

Data Analytics Lifecycle

6. **Data Aggregation and Representation:**
   - This stage is dedicated to integrating multiple datasets together to arrive at a unified view.
   - Data may be spread across multiple datasets, requiring that datasets be joined together via common fields, for example date or ID.
   - Example:



Data Analytics Lifecycle

7. **Data Analytics:**
   - This is dedicated to carrying out the actual analysis task, which typically involves one or more types of analytics. This stage can be iterative in nature.
   - Data analysis can be classified as confirmatory analysis or exploratory analysis:
     - **Confirmatory data analysis** is a *deductive* approach where the cause of the phenomenon being investigated is proposed beforehand. The proposed cause or assumption is called a hypothesis. The data is then analyzed to prove or disprove the hypothesis and provide definitive answers to specific questions.
     - **Exploratory data analysis** is an *inductive* approach that is closely associated with data mining. No hypothesis or predetermined assumptions are generated. Instead, the data is explored through analysis to develop an understanding of the cause of the phenomenon.

Data Analytics Lifecycle

8. **Data Visualization:**
   - The Data Visualization stage, is dedicated to using data visualization techniques and tools to graphically communicate the analysis results for effective interpretation by business users.
9. **Utilization of Analytics Results:**
   - This stage is dedicated to determining how and where processed analysis data can be further leveraged.
   - It is possible for the analytics results to produce "**models**" that encapsulate new insights and understandings about the nature of the **patterns and relationships** that exist within the data that was analyzed.
   - A model may look like a mathematical equation or a set of rules.

Applications

Applications

- The applications of big data span a wide range of domains including (but not limited to) homes, cities, environment, energy systems, retail, logistics, industry, agriculture, Internet of Things, healthcare, education and cybersecurity.
- We will now provide an overview of various applications of big data for some domains.

Applications

1. **Web:**
   - **Web Analytics:**
     - Web analytics deals with collection and analysis of data on the user visits on websites and cloud applications.
     - Analysis of this data can give insights about the user engagement and tracking the performance of online advertisement campaigns.
   - **Content Recommendation:**
     - Content delivery applications that serve content (such as music and video streaming applications), collect various types of data such as user search patterns and browsing history, and user ratings.
     - Such applications can leverage big data systems for recommending new content to the users based on the user preferences and interests.

Applications

2. **Financial:**
   - **Credit Risk Modeling:**
     - Banking and Financial institutions use credit risk modeling to score credit applications and predict if a borrower will fail to pay or not in the future.
   - **Fraud Detection:**
     - Banking and Financial institutions can leverage big data systems for detecting frauds such as credit card frauds, money laundering and insurance claim frauds.

Applications

3. **Internet of Things (IoT):** IoT refers to things that are connected to the Internet. The "Things" in IoT are the devices which can perform remote sensing, triggering and monitoring.
   - **Intrusion Detection:**
     - These systems use security cameras and sensors (such as PIR sensors and door sensors) to detect intrusions and raise alerts.
   - **Smart Parkings:**
     - Smart parkings are powered by IoT systems that detect the number of empty parking slots and send the information over the Internet to smart parking application back-ends.
     - These applications can be accessed by the drivers from smart-phones, tablets and in-car navigation systems.

Applications

3. **Internet of Things (IoT):**
   - **Structural Health Monitoring:**
     - Systems use a network of sensors to monitor the vibration levels in the structures such as bridges and buildings.
     - The data collected from these sensors is analyzed to assess the health of the structures.

Applications

4. **Industry:**
   - **Machine Diagnosis & Prognosis:**
     - Machine prognosis refers to predicting the performance of a machine by analyzing the data on the current operating conditions and the deviations from the normal operating conditions.
     - Machine diagnosis refers to determining the cause of a machine fault.

Applications

5. **Retail:** Retailers can use big data systems for boosting sales, increasing profitability and improving customer satisfaction.
   - **Customer Recommendations:**
     - New products can be recommended to customers based on the customer preferences, and personalized offers and discounts can be given.
     - Customers with similar preferences can be grouped and targeted campaigns can be created for customers
   - **Forecasting Demand:**
     - Big data systems can be used to analyze the customer purchase patterns and predict demand and sale volumes

Applications

6. **Environment:**
   - **Weather Monitoring**
   - **Noise Pollution Monitoring**
   - **Forest Fire Detection**
   - **River Floods Detection**
   - **Water Quality Monitoring**

Thank You

Lecture 2

Big Data Processing Techniques (MapReduce)

Dr. Lydia Wahid

Agenda

- MapReduce Definition
- MapReduce Algorithm
- MapReduce Examples

MapReduce Definition

MapReduce Definition

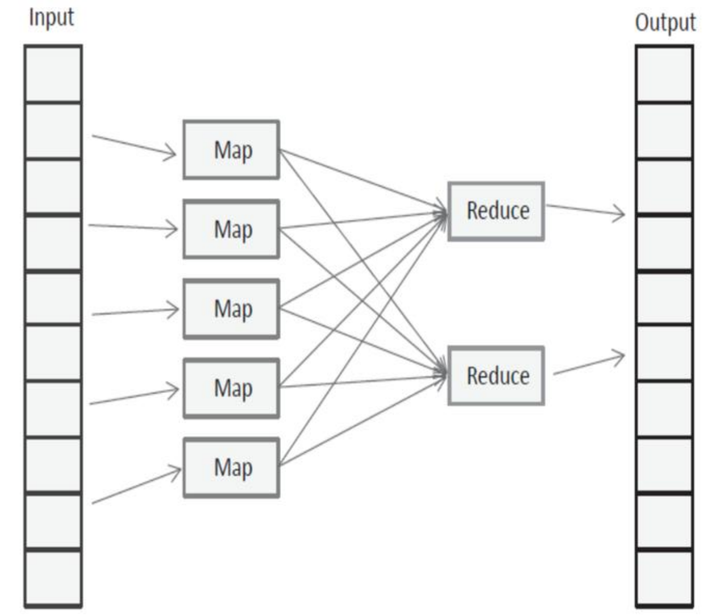
- MapReduce is a widely used Big data processing technique.
- It processes large datasets using **parallel processing** deployed over clusters of hardware.
- It is based on the principle of **divide-and-conquer**. It divides a big problem into a collection of smaller problems that can each be solved quickly.
- A **dataset is broken down** into multiple smaller parts, and operations are performed on each part independently and in parallel.
- The results from all operations are then **combined** to arrive at the result of the whole dataset.

MapReduce Definition

- Each MapReduce job is composed of a **map phase** and a **reduce phase** and each phase consists of multiple stages.
- The Map and Reduce phases run **sequentially** in a cluster.
- The Map phase is executed first then the Reduce phase.
- The output of the Map phase becomes the input of the Reduce phase.
- MapReduce does not require that the input data conform to any particular data model.

MapReduce Definition

- The following figure shows the data flow in MapReduce:

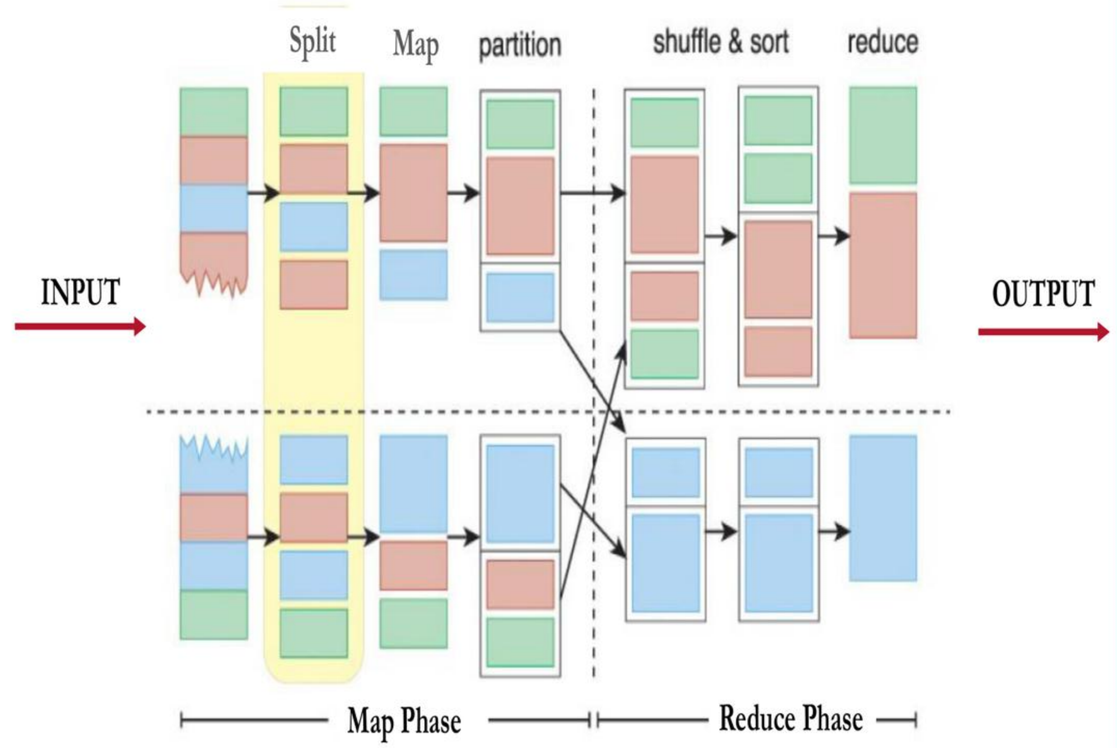


MapReduce Definition

- In MapReduce, all map and reduce tasks run in parallel.
- First of all, all map tasks are independently run.
- Meanwhile, reduce tasks wait until their respective maps are finished.
- Then, reduce tasks process their data concurrently and independently.

MapReduce Algorithm

MapReduce Algorithm



MapReduce Algorithm

- We will now apply and explain each stage on the following example:
  - **Problem Statement:**
    Count the number of occurrences of each word available in a DataSet.

Input Dataset

```
1 Red Blue Red Blue Green Red Blue Green
2 White Black
3 Red White Black
4 Orange Green
5 Red Blue Red
6 Blue Green Red Blue
7 Green White Black
```

Required Output

```
1 Black = 3
2 Blue = 6
3 Green = 5
4 Orange = 1
5 Red = 7
6 White = 3
```


MapReduce Algorithm

- Split stage:
  - Takes input DataSet and divides it into smaller Sub-DataSets called splits.
  - Each split is parsed into its constituent records as a key-value pair. The key is usually the ordinal position of the record, and the value is the actual record.
  - A common example will read a directory full of text files and return each line as a record.
  - The key-value pairs for each split are then sent to a map function (or mapper).
- By applying this stage on our example, we get the following:

MapReduce Algorithm

```
1 Red Blue Red Blue Green Red Blue Green
2 White Black
3 Red White Black
4 Orange Green
5 Red Blue Red
6 Blue Green Red Blue
7 Green White Black
```

Input Dataset

Split stage

```
1 Sub-DataSet-1
2 ---------------
3 Red Blue Red Blue Green Red Blue Green
4 White Black
5 Red White Black
```

```
1 Sub-DataSet-2
2 ---------------
3 Orange Green
4 Red Blue Red
5 Blue Green Red Blue
6 Green White Black
```

MapReduce Algorithm

- Map stage:
  - This is the map function or mapper that executes **user-defined logic**.
  - The mapper processes each key-value pair as per the user-defined logic and further generates a key-value pair as its output.
  - The output key can either be the same as the input key or a substring value from the input value, or another user-defined object.
  - Similarly, the output value can either be the same as the input value or a substring value from the input value, or another user-defined object.
  - When all records of the split have been processed, the output is a list of key-value pairs where multiple key-value pairs can exist for the same key.
- By applying this stage on our example, we get the following:

MapReduce Algorithm

```
Sub-DataSet-1
----------------
Red Blue Red Blue Green Red Blue Green
White Black
Red White Black
```

```
Sub-DataSet-2
----------------
Orange Green
Red Blue Red
Blue Green Red Blue
Green White Black
```

Map stage (mapper)

```
Sub-DataSet-1
----------------
Red = 1
Blue = 1
Red = 1
Blue = 1
Green = 1
Red = 1
Blue = 1
Green = 1
White = 1
Black = 1
Red = 1
White = 1
Black = 1
```

```
Sub-DataSet-2
----------------
Orange  = 1
Green  = 1
Red  = 1
Blue  = 1
Red  = 1
Blue  = 1
Green  = 1
Red  = 1
Blue  = 1
Green  = 1
White  = 1
Black = 1
```


MapReduce Algorithm

- Partition stage:
  - During the partition stage, if more than one reducer is involved, a partitioner divides the output from the mapper into partitions between reducer instances.
  - **All records for a particular key are assigned to the same reducer.**
  - The MapReduce algorithm guarantees a random and fair distribution between reducers while making sure that all of the same keys across multiple mappers end up with the same reducer.
- Assume here in our example, that we have only one reducer.

MapReduce Algorithm

- Shuffle and Sort stage:
  - During the first stage of the reduce task, output from all partitioners is copied across the network to the nodes running the reduce task. This is known as **shuffling**.
  - The output list of key-value pairs from each partitioner can contain the same key multiple times, so **sorting and merging** of the key-value pairs is done according to the keys so that the output contains a sorted list of all input keys and their values with the same keys appearing together.
  - This merge creates a single key-value pair per group, where key is the group key and the value is the list of all group values.
  - The way in which keys are sorted and merged can be *customized*.
- By applying this stage on our example, we get the following:

MapReduce Algorithm

```
Sub-DataSet-1
----------------
Red = 1
Blue = 1
Red = 1
Blue = 1
Green = 1
Red = 1
Blue = 1
Green = 1
White = 1
Black = 1
Red = 1
White = 1
Black = 1
```

```
Sub-DataSet-2
----------------
Orange  = 1
Green  = 1
Red  = 1
Blue  = 1
Red  = 1
Blue  = 1
Green  = 1
Red  = 1
Blue  = 1
Green  = 1
White  = 1
Black = 1
```

Shuffling and Sorting stage

```
DataSet
----------------
Black = {1,1,1}
Blue = {1,1,1,1,1,1}
Green = {1,1,1,1,1}
Orange  = {1}
Red = {1,1,1,1,1,1,1}
White = {1,1,1}
```


MapReduce Algorithm

- Reduce stage:
  - Reduce is the final stage of the reduce phase.
  - Depending on the **user-defined logic** specified in the reduce function or reducer, the reducer will either further summarize its input or will emit the output without making any changes.
  - The output key can either be the same as the input key or a substring value from the input value, or another user-defined object.
  - The output value can either be the same as the input value or a substring value from the input value, or another user-defined object.
- By applying this stage on our example, we get the following:

MapReduce Algorithm

```
DataSet
---------------
Black = {1,1,1}
Blue = {1,1,1,1,1,1}
Green = {1,1,1,1,1}
Orange = {1}
Red = {1,1,1,1,1,1,1}
White = {1,1,1}
```

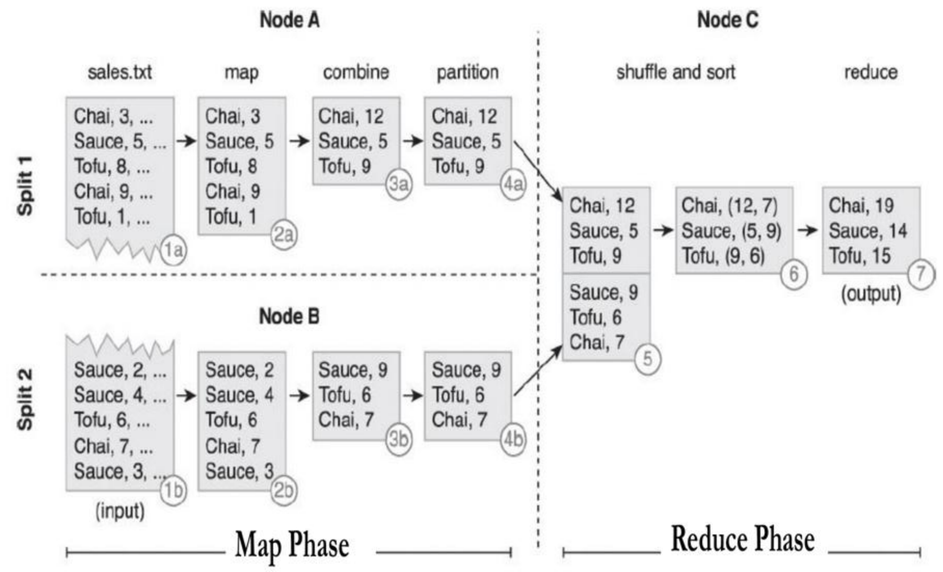
Reduce stage (reducer)

```
Black = 3
Blue = 6
Green = 5
Orange = 1
Red = 7
White = 3
```

Final Output

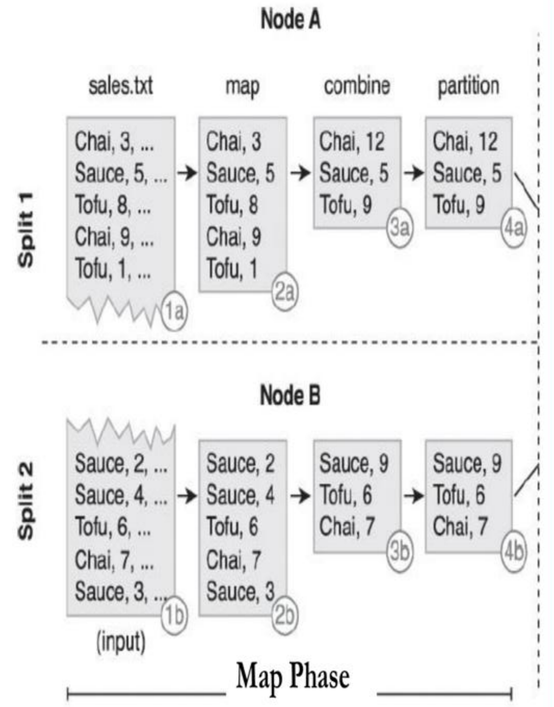
MapReduce Algorithm

- Consider another example as follows:
  - We have products information as input and we need as output the quantity of each product.



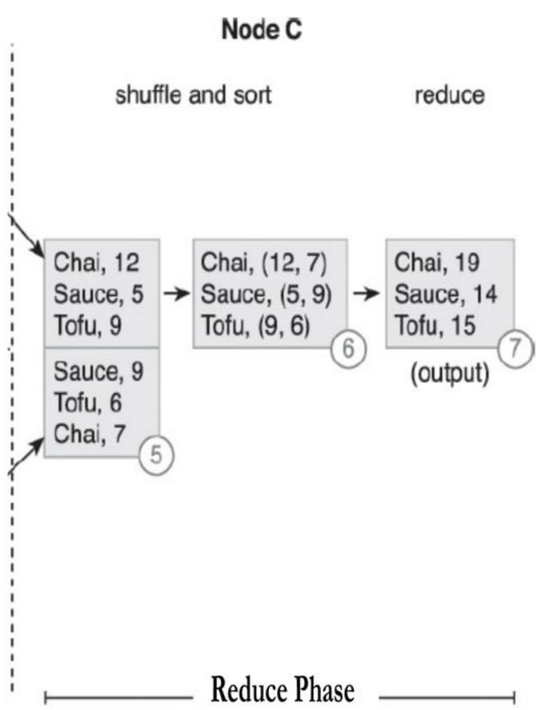
MapReduce Algorithm

1. The input (sales.txt) is divided into two splits.
2. Two map tasks running on two different nodes, Node A and Node B, extract product and quantity from the respective split's records in parallel. The output from each map function is a key-value pair where product is the key while quantity is the value.
3. The **combiner** then performs local summation of product quantities. (A combiner is essentially a reducer function that locally groups a mapper's output on the same node as the mapper.)
4. As there is only one reduce task, no partitioning is performed.



MapReduce Algorithm

5. The output from the two map tasks is then copied to a third node, Node C, that runs the shuffle stage as part of the reduce task.
6. The sort stage then groups all quantities of the same product together as a list.
7. The reduce function then sums up the quantities of each unique product in order to create the output.



MapReduce Examples

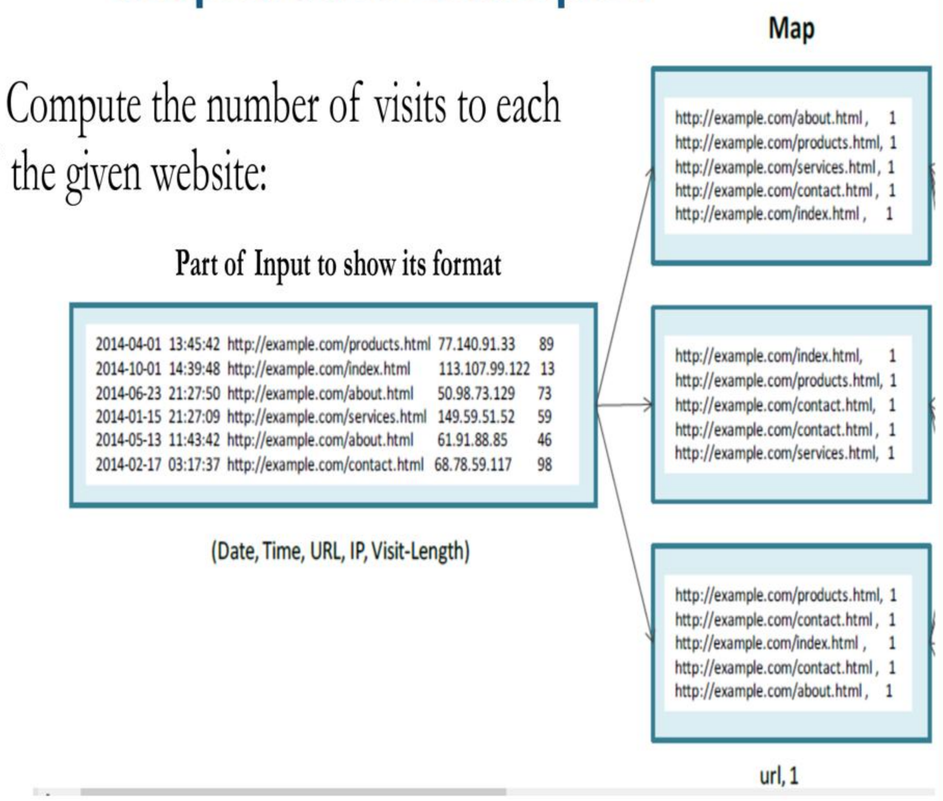
MapReduce Examples

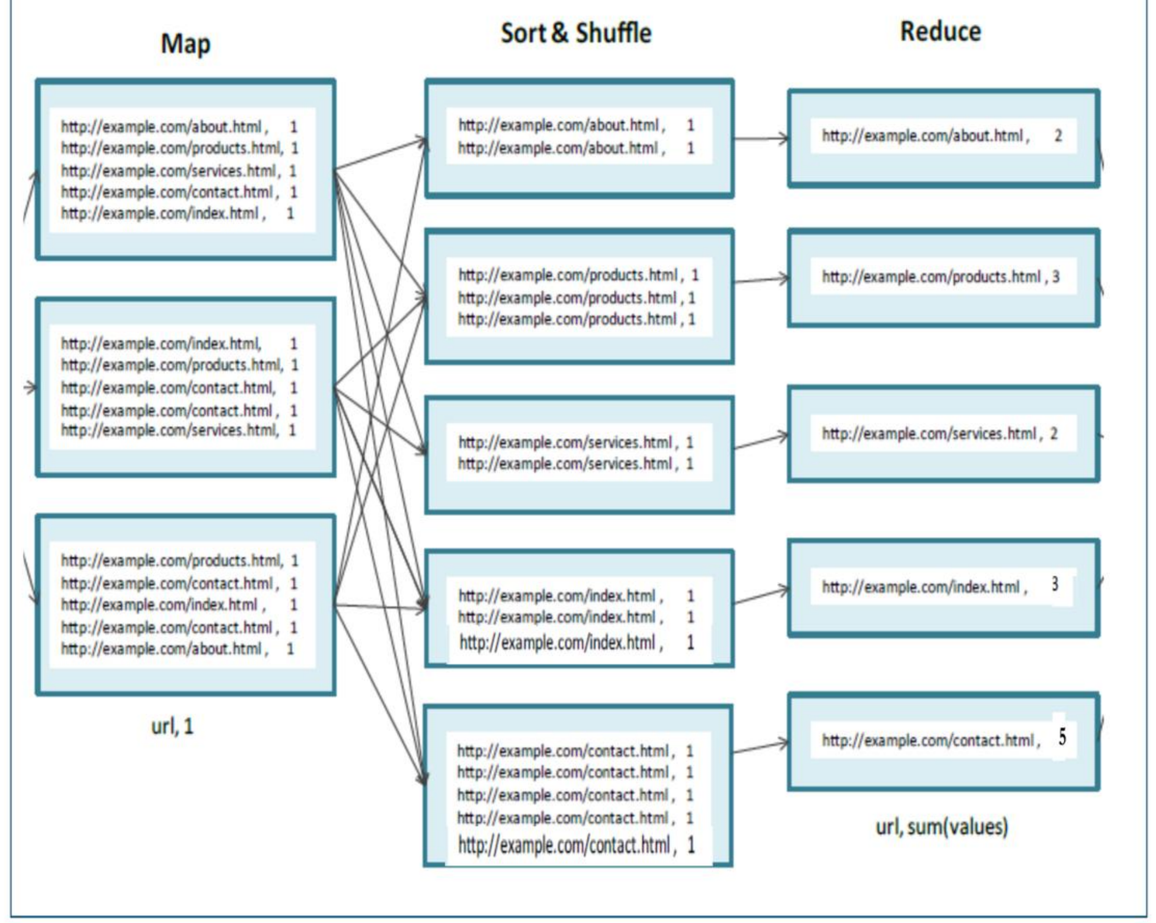
- For the examples in this section, we will use data similar to the data collected by a web analytics service that shows various statistics for page visits for a website.
- Each page has some tracking code which sends the visitor's IP address along with a timestamp to the web analytics service. The web analytics service keeps a record of all page visits and the visitor IP addresses and uses MapReduce programs for computing various statistics.
- Each visit to a page is logged as one row in the log. The log file contains the following columns:

  Date (YYYY-MM-DD), Time (HH:MM:SS), URL, IP, Visit-Length.

MapReduce Examples

1. **Count:** Compute the number of visits to each page of the given website:



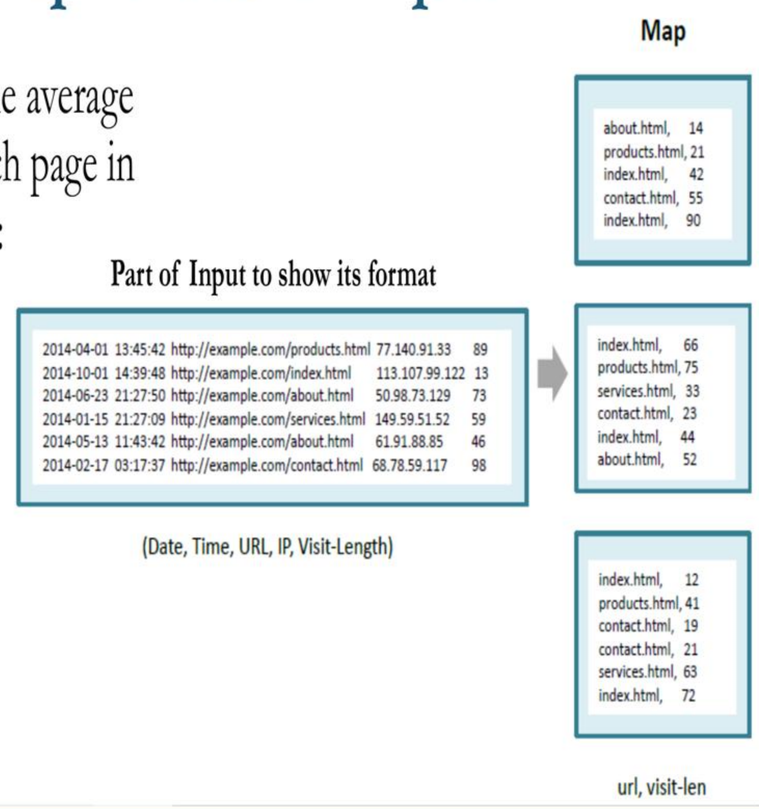


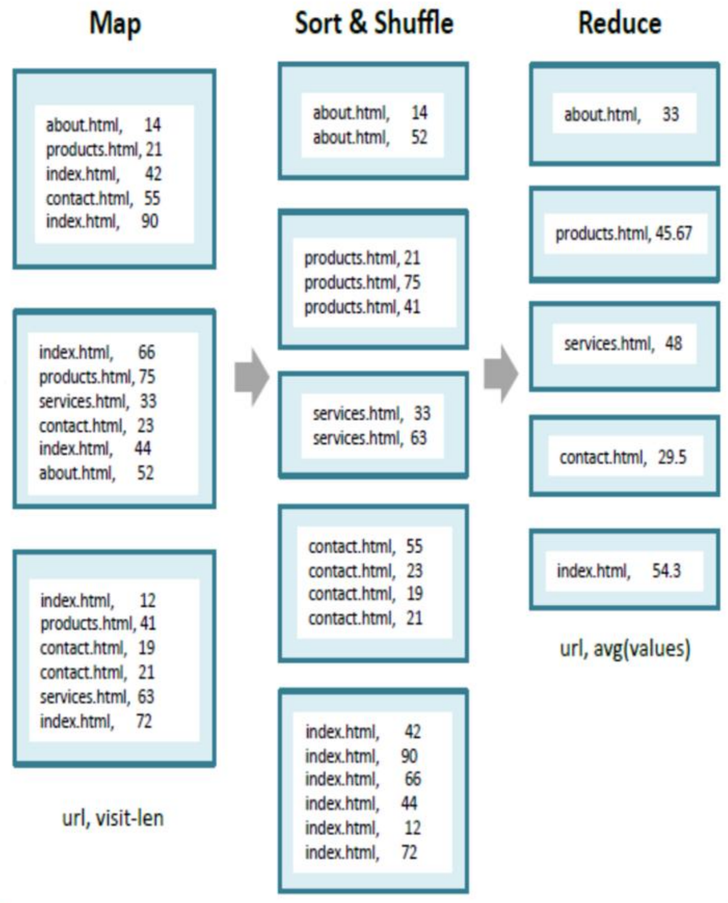
MapReduce Examples

1. **Count computation Explanation:**
   - To compute count, the mapper function emits key-value pairs where the key is the field to group-by.
   - The mapper function in this example parses each line of the input and emits key-value pairs where the key is the URL and value is '1'.
   - The reducer function receives the key-value pairs grouped by the same key and adds up the values for each group to compute count.

MapReduce Examples

2. **Average:** Find the average time spent on each page in the given website:





MapReduce Examples

2. **Average computation Explanation:**
   - To compute the average, the mapper function emits key-value pairs where the key is the field to group-by and value contains related items required to compute the average.
   - The mapper function in this example parses each line of the input and emits key-value pairs where the key is the URL and value is the visit length.
   - The reducer receives the list of values grouped by the key (which is the URL) and finds the average of these values.

MapReduce Examples

3. **Top-N:** Find the top 3 most visited pages in the given website:

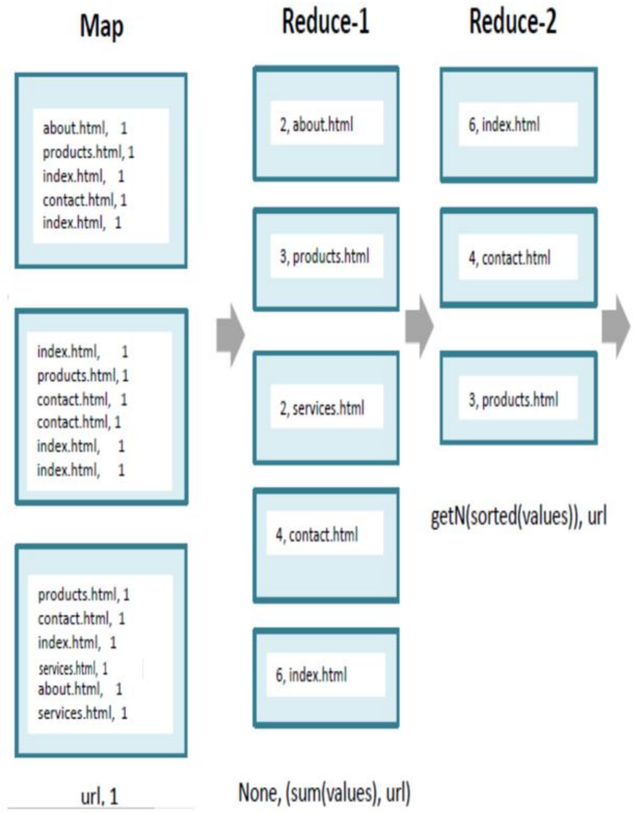
Part of Input to show its format

```
2014-04-01 13:45:42 http://example.com/products.html 77.140.91.33    89
2014-10-01 14:39:48 http://example.com/index.html    113.107.99.122  13
2014-06-23 21:27:50 http://example.com/about.html    50.98.73.129    73
2014-01-15 21:27:09 http://example.com/services.html 149.59.51.52    59
2014-05-13 11:43:42 http://example.com/about.html    61.91.88.85     46
2014-02-17 03:17:37 http://example.com/contact.html  68.78.59.117    98
```

(Date, Time, URL, IP, Visit-Length)

a. What will be the output of the shuffle and sort stage?

b. In Reduce-2, how many reducers do we have?



MapReduce Examples

3. **Top-N computation Explanation:**
   - The mapper function in this example parses each line of the input and emits key-value pairs where the key is the URL and value is '1'.
   - The reducer receives the list of values grouped by the key and sums up the values to count the visits for each page.
   - The first reducer emits None as the key and a tuple comprising of page visit count and page URL and the value.
   - The second reducer receives a list of (visit count, URL) pairs all grouped together (as the key is None). The reducer sorts the visit counts and emits top 3 visit counts along with the page URLs.
   - In this example, a two-step job was required because we need to compute the page visit counts first before finding the top 3 visited pages.

MapReduce Examples

4. **Filtering:**
   - Filter out a subset of the records based on a filtering criteria.
   - For example: filtering all page visits for the page 'contact.html' in the month of Dec 2014.

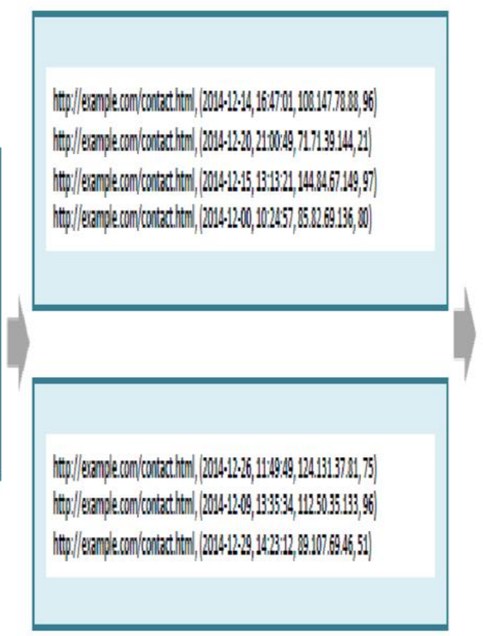
MapReduce Examples

Map

Part of Input to show its format

```
2014-04-01 13:45:42 http://example.com/products.html 77.140.91.33    89
2014-10-01 14:39:48 http://example.com/index.html    113.107.99.122  13
2014-06-23 21:27:50 http://example.com/about.html    50.98.73.129    73
2014-01-15 21:27:09 http://example.com/services.html 149.59.51.52    59
2014-05-13 11:43:42 http://example.com/about.html    61.91.88.85     46
2014-02-17 03:17:37 http://example.com/contact.html  68.78.59.117    98
```

(Date, Time, URL, IP, Visit-Length)



URL, (Date, Time, IP, Visit-Length)

MapReduce Examples

4. **Filtering computation Explanation:**
   - Filtering is useful when you want to get a subset of the data for further processing.
   - Filtering requires only a Map task.
   - Each mapper filters out its local records based on the filtering criteria in the map function.
   - The mapper function in this example parses each line of the input, extracts the month, year and page URL and emits key-value pairs if the month and year are Dec 2014 and the page URL is 'http://example.com/contact.html'.
   - The key is the URL, and the value is a tuple containing the rest of the parsed fields.

Lecture 3

Big Data Processing Frameworks (Hadoop and Spark)

Dr. Lydia Wahid

Agenda

- Basic Concepts
- Apache Hadoop
- Hadoop ecosystem
- Hadoop YARN
- Apache Spark
- Hadoop VS. Spark

Basic Concepts

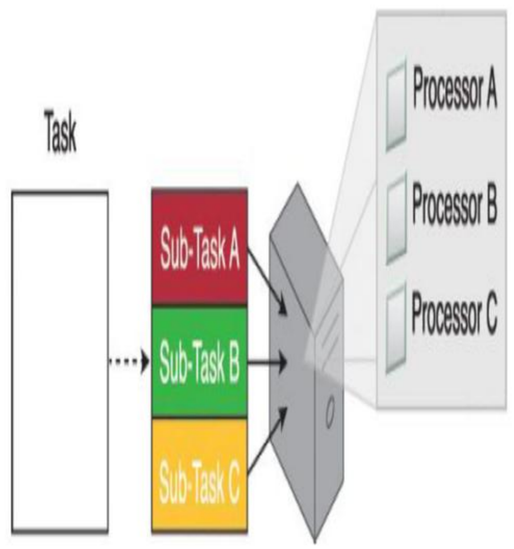
Basic Concepts

➢ **Parallel Data Processing vs Distributed Data Processing:**

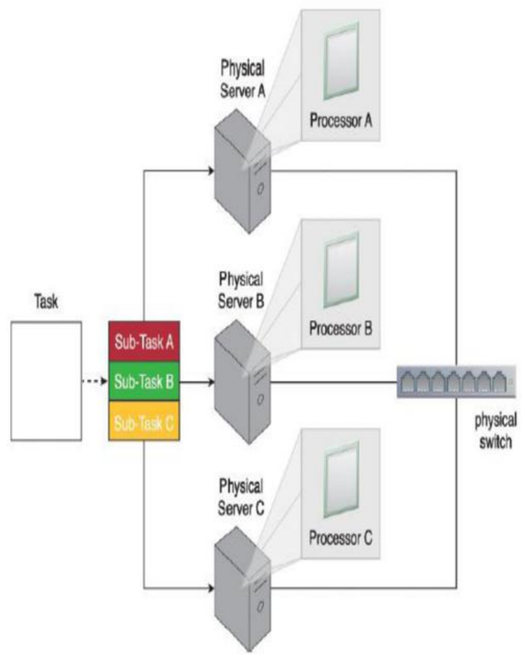
- Parallel data processing involves the **simultaneous execution** of multiple sub-tasks that collectively comprise a larger task.
- The goal is to reduce the execution time by dividing a single larger task into multiple smaller tasks that run concurrently.
- Although parallel data processing can be achieved through multiple networked machines, **it is more typically achieved within a single machine with multiple processors** or cores.
- Distributed data processing is always **achieved through physically separate machines** that are networked together as a cluster.

Basic Concepts

Parallel Data Processing



Distributed Data Processing



Basic Concepts

➢ **Batch vs Interactive vs Real-time stream Processing**

- **Batch processing** is the processing of data in groups or batches. No user interaction is required once batch processing is happening.
- **Interactive processing** means that the person needs to provide the computer with instructions while it is doing the processing.
- **Real-time stream processing** is the processing of data at the time the data is generated. Processing times can be measured in microseconds rather than in hours or days.

Apache Hadoop

Apache Hadoop

- Hadoop is an Apache open-source framework written in java that allows distributed processing of large datasets across clusters of computers.
- A cluster is a configuration of nodes that interact to perform a specific task.
- Hadoop is designed to scale up from single server to thousands of machines, each offering local computation and storage.

Apache Hadoop - Architecture

- **Hadoop Architecture:**
  - It consists of two main components:
    1. Storage (Hadoop Distributed File System)
       - MapReduce consume data from HDFS. HDFS creates multiple replicas of data blocks and distributes them on the nodes in a cluster. This distribution enables reliable and extremely rapid computations.
    2. Processing/Computation (MapReduce)
       - Capable of processing enormous data in parallel on large clusters of computation nodes. It performs two main tasks: *map* and *reduce*

Apache Hadoop - Architecture

- Hadoop Distributed File System (HDFS) is managed with the ***master-slave*** architecture included with the following components:
  - **NameNode:** This is the *master* of the HDFS system. It maintains the file system tree and the metadata for all the files and directories present in the system.
  - **DataNode:** These are *slaves* that are deployed on each machine and provide actual storage. They are responsible for serving read-and-write data requests for the clients. The internal mechanism of HDFS divides the file into one or more blocks; these blocks are stored in a set of data nodes.

Note: **HDFS** is not a **database**, but it is a distributed file system that can store huge volume of data sets across a cluster of computers to be processed

Apache Hadoop - Architecture

- MapReduce is managed with ***master-slave*** architecture included with the following components:
  - **JobTracker:** This is the *master* node of the MapReduce system, which manages the jobs and resources in the cluster. The JobTracker tries to schedule the maps to specific nodes in the cluster, ideally the nodes that have the data.
  - **TaskTracker:** These are the *slaves* that are deployed on each machine. They are responsible for running the map and reduce tasks as instructed by the JobTracker.

Apache Hadoop - Architecture

- **How MapReduce tasks are assigned to specific nodes in the cluster:**

1. Client applications submit jobs to the Jobtracker.
2. The JobTracker talks to the NameNode to determine the location of the data (DataNode)
3. The JobTracker locates TaskTracker nodes with available slots at or near the data (DataNode)
4. The JobTracker submits the work to the chosen TaskTracker nodes.
5. The TaskTracker nodes are monitored. If they do not submit signals often enough, they are considered to have failed and the work is scheduled on a different TaskTracker.
6. A TaskTracker will notify the JobTracker when a task fails. The JobTracker decides what to do then: it may resubmit the job elsewhere, it may mark that specific record as something to avoid, and it may even blacklist the TaskTracker as unreliable.
7. When the work is completed, the JobTracker updates its status.
8. Client applications can poll the JobTracker for information.

Number of Maps

- The number of maps is usually driven by the number of blocks of the input files.
- 10TB of input data and a blocksize of 128MB, will end up with 82,000 maps, unless **Configuration.set(MRJobConfig.NUM_MAPS, int)** is used to set it even higher.
- The right level of parallelism for maps seems to be around **10-100 maps per-node**, although it has been set up to 300 maps for very cpu-light map tasks.
- Block size can also be changed to adjust the number of blocks.

Number of Reducers

- In Hadoop, the default number of reducers is **one**.
- In this phase the reduce(WritableComparable, Iterable<Writable>, Context) method is called for each <key, (list of values)> pair in the grouped inputs.
- The number of reducers for the job is set by the user via **Job.setNumReduceTasks(int)**
- Increasing the number of reducers increases the framework overhead, but increases load balancing and lowers the cost of failures.
- It is legal to set the number of reduce-tasks to zero if no reduction is desired.

Apache Hadoop – HDFS Features

- **High Scalability -** HDFS is highly scalable as it can have hundreds of nodes in a single cluster.
- **Handling Huge datasets** – Due to the high scalability, HDFS can manage applications having huge datasets.
- **Distributed data storage -** This is one of the most important features of HDFS that makes Hadoop very powerful. Here, data is divided into multiple blocks and stored into nodes.
- **Hardware at data** – A requested task is done efficiently as the computation takes place near the data. This reduces the network traffic and increases the throughput, especially where huge datasets are involved.

Apache Hadoop – HDFS Features

- **Replication -** Due to some unfavorable conditions, the node containing the data may be lost. To overcome such problems, HDFS always maintains the copy of data on more than one machine.
- **Fault tolerance -** In HDFS, the fault tolerance signifies the robustness of the system in the event of failure. Due to Replication, HDFS is highly fault-tolerant that if any machine fails, the other machine containing the copy of that data automatically become active.
- **Portability -** HDFS is designed in such a way that it can be easily portable from platform to another.

Apache Hadoop – Hadoop Operation Modes

- **Hadoop can run in 3 different modes:**

1. Standalone Mode (**Single** machine **Single** process):
   By default, Hadoop is configured to run in a non-distributed mode. It runs as a single Java process. Instead of HDFS, this mode utilizes the local file system. This mode is useful for debugging.
2. Pseudo-Distributed Mode (**Single** machine **Multiple** processes):
   Hadoop can also run on a single machine in a Pseudo Distributed mode. In this mode, each Hadoop process runs as separate java process. Here HDFS is utilized for input and output.
3. Fully Distributed Mode (**Multiple** machines **Multiple** processes):
   This is the production mode of Hadoop. In this mode, Hadoop is distributed across multiple machines. Therefore, separate Java processes are present. This mode offers fully distributed computing capability, reliability, fault tolerance and scalability.

Hadoop ecosystem

Apache Hadoop – Hadoop ecosystem

- **Apache Hive**
  - It is a software developed by Facebook that facilitates reading, writing, and managing large datasets residing in distributed storage using SQL.
  - It allows users to fire queries in SQL-like languages, such as **HiveQL**.

- **Apache Pig:**
  - It is platform for creating programs that run on Apache Hadoop. The language for this platform is called **Pig Latin**.
  - Apache Pig has been developed by Yahoo. Currently, Yahoo and Twitter are the primary users of Pig.

Apache Hadoop – Hadoop ecosystem

- **Apache Hbase:**
  - It is the Hadoop database, a distributed, scalable, big data store. This allows random, real-time read/write access to Big Data. Apache HBase is an open-source, distributed, non-relational database modeled after Google's Bigtable.
  - The following are some companies using HBase: Yahoo, Twitter, and stumble upon (This is a personalized recommender system, realtime data storage, and data analytics platform).

Apache Hadoop – Hadoop ecosystem

- **Apache Impala:**
  - With Impala, you can query data, whether stored in HDFS or Apache HBase – including SELECT, JOIN, and aggregate functions – in real time.
  - Impala directly access the data through a specialized distributed query engine. The result is order-of-magnitude faster performance than Hive
- **Apache Sqoop:**
  - Apache Sqoop is a mutual data tool for importing data from the relational databases to Hadoop HDFS and exporting data from HDFS to relational databases.
  - It works together with most modern relational databases, such as Microsoft SQL Server, MySQL, and Oracle.

Apache Hadoop – Hadoop ecosystem

- **Mahout:**
  - It is a popular data mining library. It includes the most popular data mining scalable machine learning algorithms. Also, it is a scalable machine-learning library.
  - The following are some companies that are using Mahout: Amazon, Twitter, Yahoo, and LucidWorks Big Data (This is an analytics firm, which uses Mahout for clustering, duplicate document detection, phase extraction, and classification).

Apache Hadoop – Hadoop ecosystem

- **Apache Solr:**
  - It is an open-source enterprise search platform.
  - Solr is highly reliable, scalable and fault tolerant, providing distributed indexing, replication and load-balanced querying, automated failover and recovery, centralized configuration and more.
  - This allows building web application with powerful search capabilities.
  - Solr powers the search and navigation features of many of the world's largest internet sites

Apache Hadoop – Hadoop ecosystem

- **Apache Zookeeper:**
  - It is a service that enables highly reliable distributed coordination.
  - It is a centralized service for maintaining configuration information, naming, and providing distributed synchronization.
- …..and others

Hadoop YARN

Hadoop YARN

- The JobTracker in v1.0 is the single master that allocates resources for applications, performs scheduling for demand and also monitors the jobs of processing in the system. (master for resources + processing)
- YARN stands for "***Yet Another Resource Negotiator***". It was introduced in Hadoop version 2.0.
- **The fundamental idea of YARN is to split up the functionalities of resource management and job scheduling/monitoring.**

Hadoop YARN

- Thus, YARN is now responsible for Resource Management and Job scheduling.
- Through its various components, YARN can dynamically allocate various resources and schedule the application processing.
- The idea is to have a global ResourceManager (*RM*) and per-application ApplicationMaster (*AM*). An application is either a single job or a DAG of jobs.

Hadoop YARN - Architecture

- The main components of YARN architecture include:

1. **Resource Manager:** It is the master of YARN and is responsible for **resource assignment and management** among all the applications.
   - Whenever it receives a processing request, it forwards it to the corresponding node manager and allocates resources for the completion of the request accordingly.
   - The ResourceManager has two main components: *Scheduler* and *ApplicationsManager*.

Hadoop YARN - Architecture

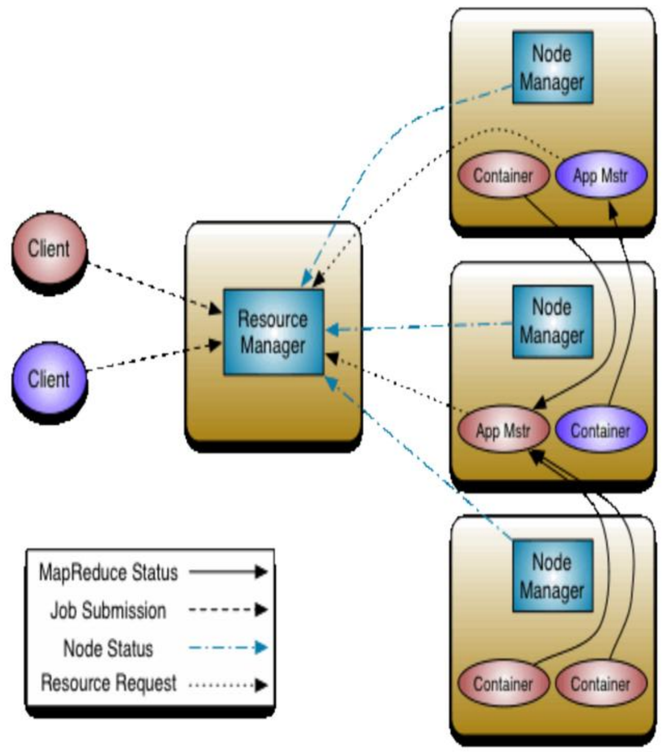
- The main components of YARN architecture include:

1. **Resource Manager:**
   - **ApplicationsManager** is responsible for accepting job-submissions, negotiating the first container for executing the application specific ApplicationMaster and provides the service for restarting the ApplicationMaster container on failure.
   - **Scheduler** is responsible for allocating resources to the various running application. It performs its scheduling function based on the resource requirements of the applications

Hadoop YARN - Architecture

2. **Container:** In the Container, there are the physical resources like a disk, CPU cores, RAM.
3. **Application Master:** An application is a single job submitted to a framework. The application master is responsible for **negotiating resources** with the resource manager, **executing, tracking the status**, and **monitoring progress** of a single application.
4. **Node manager:** It **sends each node's health status** to the Resource Manager, stating if the node process has finished working with the resource. Node manager is also responsible for **monitoring resource usage** by individual Container and reporting it to the Resource manager.

Hadoop YARN - Architecture



Hadoop YARN - Features

- **Multi-tenancy:** YARN has allowed access to multiple data processing engines such batch, interactive, and real-time stream processing.
- **Scalability:** The scheduler in Resource manager of YARN architecture allows Hadoop to extend and manage thousands of nodes and clusters.
- **Cluster utilization:** YARN allocates all cluster resources in an efficient and dynamic manner, which leads to better utilization.
- **Compatibility:** YARN supports the existing map-reduce applications without disruptions thus making it compatible with Hadoop 1.0 as well.

Hadoop Version 2.0 vs Version 1.0

Criteria	Version 2.0	Version 1.0
Components	• HDFS • MapReduce • YARN	• HDFS • MapReduce
Suitable for	MapReduce and non-MapReduce applications	Only MapReduce applications
Managing cluster resource	Done by YARN	Done by JobTracker
Cluster resource optimization	Excellent due to central resource management	Average due to fixed Map and Reduce slots

Apache Spark

Apache Spark

- Apache Spark is a data processing framework that can quickly perform processing tasks on very large data sets, and can also distribute data processing tasks across multiple computers.
- Apache Spark is a prime example of how YARN enables customers to build a real-time analytics platform.
- Spark also takes some of the programming burdens of these tasks off the shoulders of developers with an easy-to-use API that abstracts away much of the work of distributed computing and big data processing.

Apache Spark - Features

- **In-memory computing** – The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.
- **Speed** – Spark helps to run an application in Hadoop cluster faster when running on disk and even faster when run in memory. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages** – Spark provides built-in APIs in Java, R, Python and Scala.
- **Advanced Analytics** – Spark not only supports Map and reduce, but also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

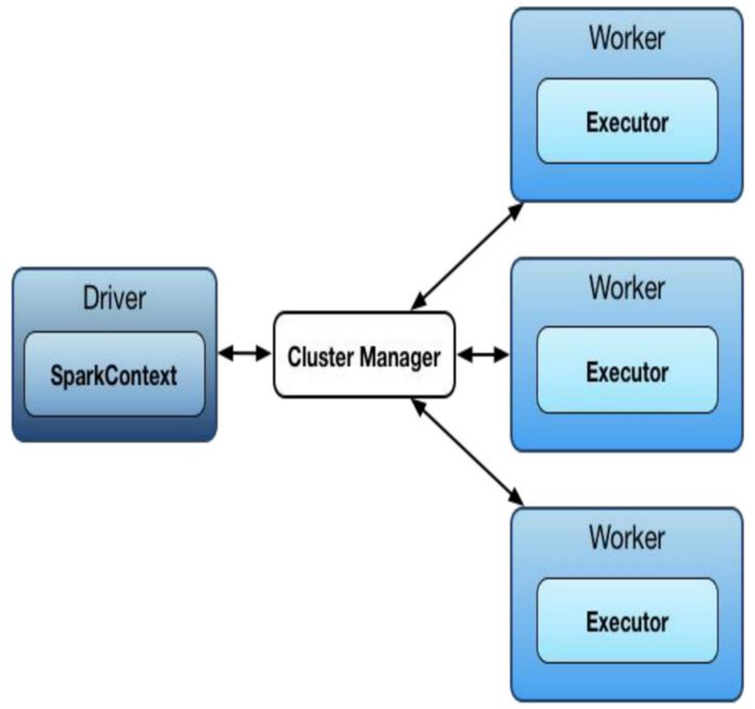
Apache Spark - Data structure

- **Resilient Distributed Datasets (RDD)** is a fundamental data structure of Spark. It is a distributed collection of objects.
- RDDs are the building blocks of any Spark application. RDDs Stands for:
  - ***Resilient:*** Fault tolerant and is capable of rebuilding data on failure
  - ***Distributed:*** Distributed data among the multiple nodes in a cluster
  - ***Dataset:*** Collection of partitioned data with values
- Each dataset is divided into logical partitions, which may be computed on different nodes of the cluster.

Apache Spark - Architecture

- At a fundamental level, an Apache Spark follows a master/slave architecture with two main components: a *driver* and *workers*.
  - **Driver (Master Node):** It is the entry point that runs the main () function of the application and is responsible for the translation of spark user code into actual spark jobs executed on the cluster.
  - **Workers (Slave Nodes):** They contain the *executers* that run tasks assigned to them. These tasks are executed on the partitioned RDDs in the worker node. Executors perform all the data processing.
- There is also the **Cluster Manager** which is an external service responsible for acquiring resources on the spark cluster and allocating them to a spark job.

Apache Spark - Architecture



Spark context is a gateway to all the Spark functionalities. It is similar to your database connection.

Apache Spark - RDD

- RDD is a read-only, partitioned collection of records. RDD is a fault-tolerant collection of elements that can be operated on in parallel.
- There are two ways to create RDDs:
  - **Parallelizing** an existing collection in your driver program.
  - **Referencing a dataset** in an external storage system, such as a shared file system, HDFS, HBase, or other data sources.

Apache Spark - RDD

- Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations.
- With RDDs, you can perform two types of operations:
  - **Transformations:** They are the operations that are applied to create a new RDD. They are **lazy**, their result RDD is not immediately computed.
  - **Actions:** They are applied on an RDD to instruct Apache Spark to apply computation and pass the result back to the driver. They are **eager**, their result is immediately computed.

Apache Spark - RDD

- **Transformations examples:**
  - **map**(*func*): Return a new distributed dataset formed by passing each element of the source through a function func.
  - **filter**(*func*): Return a new dataset formed by selecting those elements of the source on which func returns true.
  - **intersection**(*otherDataset*): Return a new RDD that contains the intersection of elements in the source dataset and the argument.
  - **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 func.

Apache Spark - RDD

- **Actions examples:**
  - **reduce**(*func*): Aggregate the elements of the dataset using a function func.
  - **count**(): Return the number of elements in the dataset.
  - **first**(): Return the first element of the dataset.
  - **take**(*n*): Return an array with the first n elements of the dataset.
  - **takeOrdered**(*n, [ordering]*): Return the first n elements of the RDD using either their natural order or a custom comparator.

Apache Spark - Workflow

- **STEP 1:** The client submits spark user application code. When an application code is submitted, the driver implicitly converts user code into a logically *directed acyclic graph* called ***DAG.*** At this stage, it also performs optimizations.

- **STEP 2:** After that, it converts the logical graph called DAG into physical execution plan with many stages. After converting into a physical execution plan, it creates physical execution units called tasks under each stage. Then the tasks are bundled and sent to the cluster.

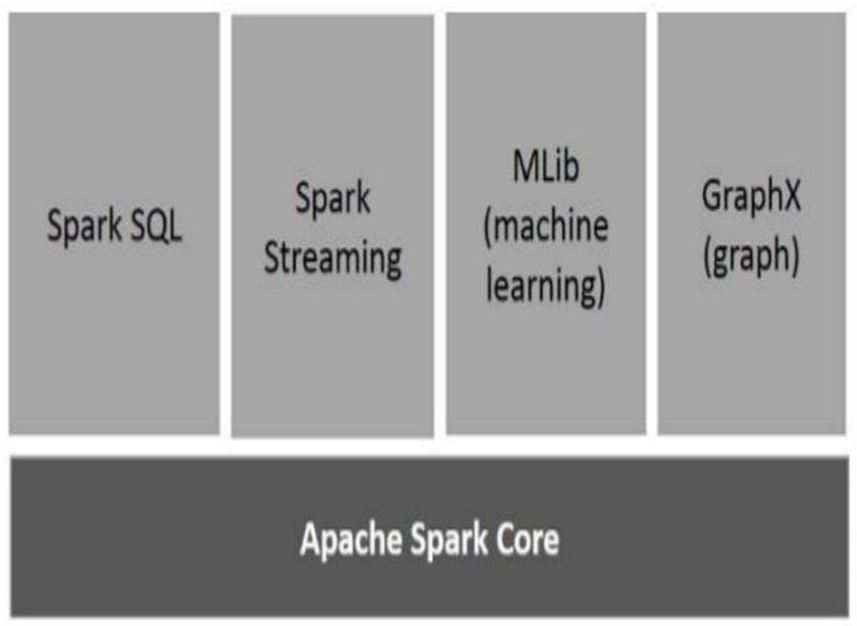
Apache Spark - Workflow

- **STEP 3:** Now the driver talks to the cluster manager and negotiates the resources. Cluster manager launches executors in worker nodes on behalf of the driver. At this point, the driver will send the tasks to the executors based on data placement. The driver will have a complete view of executors that are executing the task.

- **STEP 4:** During the course of execution of tasks, driver will monitor the set of executors that runs. Driver node also schedules future tasks based on data placement.

Apache Spark - ecosystem

- The following illustration depicts the different components of Spark:



Apache Spark - ecosystem

- Apache Spark Core:
  - Spark Core is the underlying general execution engine for spark platform that all other functionality is built upon. It provides In-Memory computing and referencing datasets in external storage systems.
- Spark SQL:
  - Spark SQL enables users to run SQL queries. Alongside standard SQL support, Spark SQL provides a standard interface for reading from and writing to other datastores including JSON, HDFS, Apache Hive and others.

Apache Spark - ecosystem

- Spark Streaming:
  - Spark Streaming extended the Apache Spark concept of batch processing into streaming by breaking the stream down into a continuous series of microbatches, which could then be manipulated using the Apache Spark API.
- MLlib (Machine Learning Library)
  - Machine learning library delivers both efficiencies as well as the high-quality algorithms. It is capable of in-memory data processing, that improves the performance of iterative algorithm drastically.
- GraphX
  - Spark GraphX is the graph computation engine built on top of Apache Spark that enables to process graph data at scale.

Hadoop VS Spark

Hadoop VS Spark

	Hadoop	Spark
Processing Speed & Performance	Hadoop reads and writes from a disk, thus **slowing down** the processing speed.	Spark reduces the number of read/write cycles to disk and stores intermediate data in memory, hence **faster-processing** speed.
Usage	Hadoop is designed to handle **batch** processing efficiently.	Spark is designed to handle **real-time** data efficiently.
Fault Tolerance	Fault-tolerance achieved by **replicating blocks of data.** If a node goes down, the data can be found on another node	Fault-tolerance achieved by **storing chain of transformations.** If data is lost, the chain of transformations can be recomputed on the original data.

Hadoop VS Spark

	Hadoop	Spark
Use Cases	• Processing big data sets in environments where **data size exceeds available memory** • Building data analysis infrastructure with a **limited budget** • Completing jobs that are **not time-sensitive** • **Historical and archive** data analysis	• Dealing with chains of parallel operations by using **iterative algorithms** • Analyzing **stream data** in real time • **Graph-parallel processing** to model data • **All ML** applications

Thank You

Lecture 4

Big Data Predictive Analytics

Dr. Lydia Wahid

Agenda

- Introduction
- K-Nearest Neighbor
- Applying KNN on Big Data
- Naïve Bayes classifier
- Applying Naïve Bayes classifier on Big Data
- Performance Evaluation of classifiers

Introduction

Introduction

- **Predictive Analytics** use **Predictive Data Mining** techniques which use **Supervised Machine Learning** techniques.

Predictive Analytics → use → **Predictive Data Mining** → use → **Supervised Machine Learning**

Introduction

- *Classification* is an instance of Supervised Machine Learning and is widely used for prediction purposes.
- In classification, a classifier is given a set of examples that are already classified (i.e. given a class label), and from these examples, the classifier learns to assign a label to unseen examples.
- Examples of classification problems include:
  - Given an email, classify if it is spam or not.
  - Given a handwritten character, classify it as one of the known characters.

Introduction

- **Examples of Classification Techniques:**
  - **K-Nearest Neighbor (KNN)**
  - **Naïve Bayes**
  - **Decision Trees (DT)**
  - **Support Vector Machines (SVM)**
  - **Neural Networks**

K-Nearest Neighbor

K-Nearest Neighbor

- K-nearest neighbors is an algorithm that stores all available cases and classifies new cases based on a **similarity measure** (e.g., distance functions).
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.

K-Nearest Neighbor

- The K-NN working can be explained on the basis of the below algorithm:

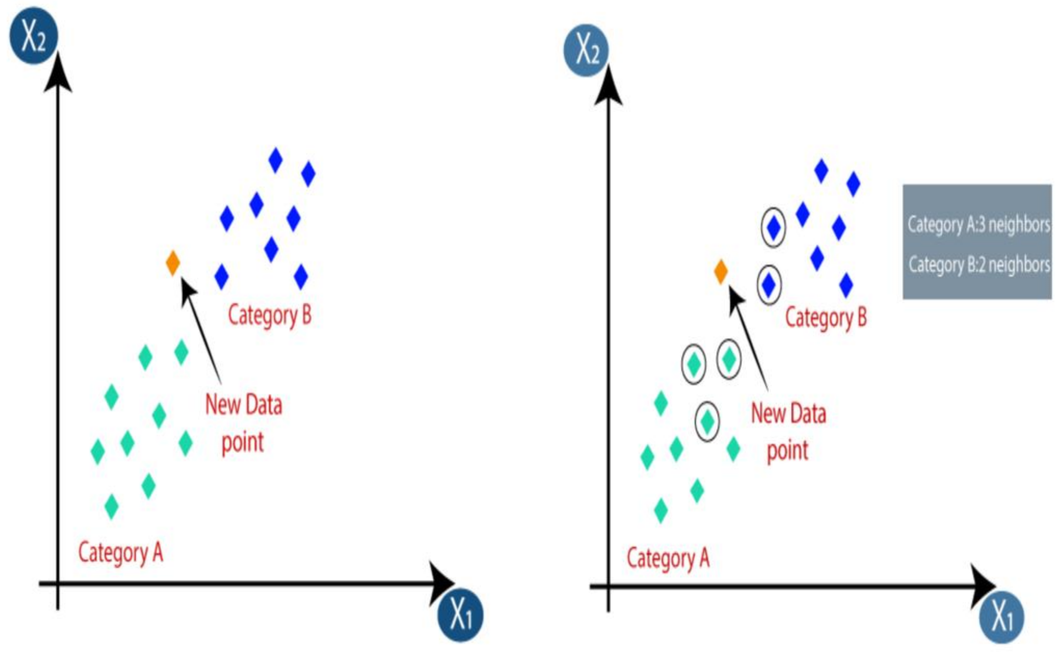
Step 1 − For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 − Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 − For each point in the test data do the following −

- **3.1** − Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance.
- **3.2** − Now, based on the distance value, sort them in ascending order.
- **3.3** − Next, it will choose the top K rows from the sorted array.
- **3.4** − Now, it will assign a class to the test point based on most frequent class of these rows.

K-Nearest Neighbor



K-Nearest Neighbor

- **Distance functions:**

1. Euclidean Distance:

$$D = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

n is the number of features

2. Manhattan Distance:

$$D = \sum_{i=1}^{n}|x_i - y_i|$$

n is the number of features

K-Nearest Neighbor

- **Distance functions:**

3. Hamming Distance:

- It is a measure of the number of instances in which corresponding symbols are different in two strings of equal length. It is suitable for categorial features.

$$D_H = \sum_{i=1}^{n}|x_i - y_i|$$

n is the number of features

$$x = y \Rightarrow D = 0$$

$$x \neq y \Rightarrow D = 1$$

K-Nearest Neighbor

➢Example:

Height (in cms)	Weight (in kgs)	T Shirt Size
158	58	M
158	59	M
158	63	M
160	59	M
160	60	M
163	60	M
163	61	M
160	64	L
163	64	L
165	61	L
165	62	L
165	65	L

New customer has height 161cm and weight 61kg.

What is his T Shirt Size?

K-Nearest Neighbor

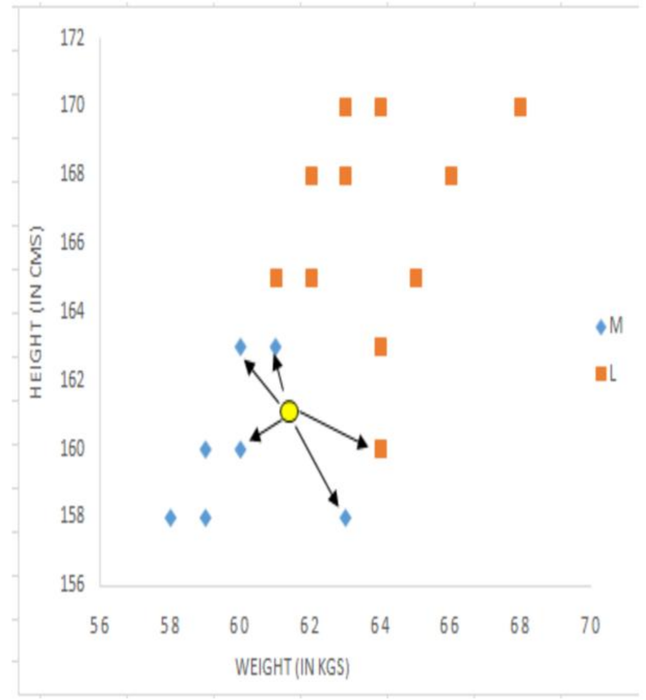
➢Example:

Height (in cms)	Weight (in kgs)	T Shirt Size	Distance	
158	58	M	4.2	
158	59	M	3.6	
158	63	M	3.6	
160	59	M	2.2	3
160	60	M	1.4	1
163	60	M	2.2	3
163	61	M	2.0	2
160	64	L	3.2	5
163	64	L	3.6	
165	61	L	4.0	
165	62	L	4.1	
165	65	L	5.7	

Euclidean Distance is used.
For K=5,
T shirt Size=M

K-Nearest Neighbor

➢Example:



K-Nearest Neighbor

What are the limitations of KNN?

K-Nearest Neighbor

➢Normalization and Standardization:

- When independent variables in training data are measured in different units, it is important to **scale the variables** before calculating distance.
- For example, if one variable is based on height in cms, and the other is based on weight in kgs then **height will influence more** on the distance calculation.
- Scaling the variables can be done by any of the following methods:

$$Xs = \frac{X - min}{max - min}$$

Normalization

$$Xs = \frac{X - mean}{s.d.}$$

Standardization

K-Nearest Neighbor

➢Handling categorial features:

- Hamming Distance can be used
- Can assign a number to each category

K-Nearest Neighbor

- How to find best K value?
  - **Cross-validation** is a way to find out the **optimal K value**. It estimates the validation error rate by holding out a subset of the training set from the model building process.
  - Cross-validation (let's say 10 fold validation) involves randomly dividing the **training set** into 10 groups, or folds, of approximately equal size. 90% data is used to train the model and remaining 10% to validate it. The error rate is then computed on the 10% validation data. This procedure repeats 10 times each time with a different fold. It results to 10 estimates of the validation error which are then averaged out.
  - The process is repeated for different values of K. The value of K that yields the smallest average error is selected.

Applying KNN on Big Data

Applying KNN on Big Data

- Despite the promising results shown by the k-NN in a wide variety of problems, it lacks scalability to address Big datasets.
- The main problems found to deal with large-scale data are:
  - **Runtime:** The complexity of the traditional k-NN algorithm is $O((n \cdot D))$, where n is the number of instances and D the number of features.
  - **Memory consumption:** For a rapid computation of the distances, the k-NN model may normally require to store the training data in memory. When TR is too big, it could easily exceed the available RAM memory.

Applying KNN on Big Data

- These drawbacks motivate the use of Big Data techniques to distribute the processing of KNN over a cluster a nodes.
- A MapReduce-based approach for k-Nearest neighbor classification can be applied.
- This allows us to simultaneously classify large amounts of unseen cases (test examples) against a big (training) dataset.

Applying KNN on Big Data

- First, the training data will be divided into multiple splits.
- The **map phase** will determine the k-nearest neighbors in the different splits of the data.
- As a result of each map, the k nearest neighbors together with their computed distance values will be emitted to the reduce phase.

Applying KNN on Big Data

- Afterwards, the **reduce phase** will compute the definitive neighbors from the list obtained in the map phase.
- The reduce phase will determine which are the final k nearest neighbors from the list provided by the maps.
- This parallel implementation provides the exact classification rate as the original k-NN model.

Naïve Bayes classifier

Naïve Bayes

- Naïve Bayes is a probabilistic classification method based on Bayes' theorem (or Bayes' law).
- Bayes' theorem gives the **relationship between the probabilities of two events and their conditional probabilities**.
- A naïve Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of other features.

Naïve Bayes

- The input variables are generally **categorical**, but variations of the algorithm can accept continuous variables.
- There are also ways to convert continuous variables into categorical ones. This process is often referred to as the ***discretization of continuous variables***.
- The output typically includes a class label and its corresponding probability score.

Naïve Bayes: Bayes' Theorem

- The ***conditional probability*** of event C occurring, given that event A has already occurred, is denoted as $P(C|A)$, which can be found using the following equation:

$$P(C|A) = \frac{P(A \cap C)}{P(A)}$$

$$P(C|A) = \frac{P(A|C) \cdot P(C)}{P(A)}$$

where C is the class label $C \in \{c1, c2, \ldots cn\}$ and A is the observed attributes $A = \{a1, a2, \ldots am\}$

Naïve Bayes: Bayes' Theorem

- A more general form of Bayes' theorem assigns a classified label to an object with multiple attributes $A = \{a1, a2, \ldots am\}$ such that the label corresponds to the largest value of $P(ci|A)$.

$$P(c_i|A) = \frac{P(a_1, a_2, \ldots, a_m|c_i) \cdot P(c_i)}{P(a_1, a_2, \ldots, a_m)}, i = 1, 2, \ldots n$$

Naïve Bayes classifier

- With two simplifications, Bayes' theorem can be extended to become a naïve Bayes classifier.
- **The first simplification** is to use the conditional independence assumption. That is, each attribute is conditionally independent of every other attribute given a class label c*i*.
- **The second simplification** is to ignore the denominator $P(a1, a2, \ldots am)$ because it appears in the denominator for all values of *i*, removing the denominator will have no impact on the relative probability scores and will simplify calculations.

Naïve Bayes classifier

- Naïve Bayes classification applies the two simplifications mentioned earlier and, as a result, $P(ci \mid a1, a2,..am)$ is proportional to the product of $P(aj \mid ci)$ times $P(ci)$.

$$P(c_i|A) \propto P(c_i) \cdot \prod_{j=1}^{m} P(a_j|c_i) \qquad i = 1,2,\ldots n$$

Naïve Bayes classifier

- Building a naïve Bayes classifier requires knowing certain statistics, all calculated from the training set.
- The first requirement is to collect the probabilities of all class labels, $P(ci)$.
- The second thing the naïve Bayes classifier needs to know is the conditional probabilities of each attribute aj given each class label ci, namely $P(aj \mid ci)$. For each attribute and its possible values, computing the conditional probabilities given each class label is required.

Naïve Bayes classifier

- For a given attribute assume it can have the following values {x,y,z} and assume that we have two class labels {c1 and c2}.
- Then the following probabilities need to be computed:
  - P(x | c1)
  - P(x | c2)
  - P(y | c1)
  - P(y | c2)
  - P(z | c1)
  - P(z | c2)

Naïve Bayes classifier

- After that, the naïve Bayes classifier can be tested over the testing set.
- For each record in the testing set, the naïve Bayes classifier assigns the classifier label ci that maximizes:

$$P(c_i) \cdot \prod_{j=1}^{m} P\left(a_j|c_i\right)$$

where:
- m is the number of features (dimensions)
- i is the index of class labels
- j is the index of features (dimensions)
- a1 is the value of the first feature in the test record
- a2 is the value of the second feature in the test record…….

Applying Naïve Bayes classifier on Big Data

Applying Naïve Bayes classifier on Big Data

- Applying MapReduce with Naïve Bayes classifier significantly decreases computation times allowing its application on Big Data problems.
- First, the training data will be divided into multiple splits.
- During the map phase, each map processes a single split, and computes statistics of the input data.
- For each attribute, the map outputs a <Key, Value> pair, where
  - Key is the class label,
  - Value is {AttributeValue, the frequency of attribute value within that class label}

Applying Naïve Bayes classifier on Big Data

- In the **reduce phase**, the reduce function aggregates the number of each attribute value within each class label.
- For each attribute, the reduce function outputs a <Key, Value> pair, where:
  - Key is the class label,
  - Value is {AttributeValue, Σi the frequency of attribute value within that class label}

  where i is the number of mappers.

Performance Evaluation of classifiers

Performance Evaluation of classifiers

- A ***confusion matrix*** is a specific table layout that allows visualization of the performance of a classifier.
- The following figure shows the confusion matrix for a two-class classifier:

		Predicted Class	
		Positive	Negative
Actual Class	Positive	True Positives (TP)	False Negatives (FN)
	Negative	False Positives (FP)	True Negatives (TN)

Performance Evaluation of classifiers

- ***True positives*** (TP) are the number of positive instances the classifier correctly identified as positive.
- ***False positives*** (FP) are the number of instances in which the classifier identified as positive but in reality are negative.
- ***True negatives*** (TN) are the number of negative instances the classifier correctly identified as negative.
- ***False negatives*** (FN) are the number of instances classified as negative but in reality are positive.

Performance Evaluation of classifiers

- The ***accuracy*** (or the ***overall success rate***) is a metric defining the rate at which a model has classified the records correctly.
- It is defined as the sum of TP and TN divided by the total number of instances, as shown in the following equation:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\%$$

Performance Evaluation of classifiers

- The ***false positive rate*** (FPR) shows what percent of negatives the classifier marked as positive.
- The FPR is also called the ***false alarm rate*** or the ***type I error rate .***
- FPR is computed as follows:

$$FPR = \frac{FP}{FP + TN}$$

Performance Evaluation of classifiers

- ***Precision*** is the percentage of instances marked positive that really are positive. It is computed as follows:

$$Precision = \frac{TP}{TP + FP}$$

- ***Recall*** is the percentage of positive instances that were correctly identified. It is also called ***true positive rate*** (TPR). It is computed as follows:

$$TPR\ (or\ Recall) = \frac{TP}{TP + FN}$$

Performance Evaluation of classifiers

- ***F1-score*** is the harmonic mean of the precision and recall:

$$F_1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

Performance Evaluation of classifiers – Multi classes

- **Micro-average and Macro-average:**
  - The **micro-average** precision and recall scores are calculated from the sum of classes' true positives (TPs), false positives (FPs), and false negatives (FNs) of the model.
  - The **macro-average** precision and recall scores are calculated as arithmetic mean (or weighted mean) of individual classes' precision and recall scores.
  - The **macro-average** F1-score is calculated as arithmetic mean (or weighted mean) of individual classes' F1-score.

Performance Evaluation of classifiers – Multi classes

- Exercise:

No	Actual	Predicted	Match
1	Airplane	Airplane	✓
2	Car	Boat	X
3	Car	Car	✓
4	Car	Car	✓
5	Car	Boat	X
6	Airplane	Boat	X
7	Boat	Boat	✓
8	Car	Airplane	X
9	Airplane	Airplane	✓
10	Car	Car	✓

Performance Evaluation of classifiers – Multi classes

- Exercise:

	Airplane	Boat	Car
Airplane	2	1	0
Boat	0	1	0
Car	1	2	3

Confusion matrix

Label	True Positive (TP)	False Positive (FP)	False Negative (FN)
Airplane	2	1	1
Boat	1	3	0
Car	3	0	3

TP, FP, FN calculated from the confusion matrix

Performance Evaluation of classifiers – Multi classes

- Exercise:
  - Calculate micro-average, macro-average, and weighted macro-average of precision, recall and F1-score.

Lecture 5

Big Data Descriptive Analytics

Dr. Lydia Wahid

Agenda

- Introduction
- Clustering
- Applying Clustering on Big Data
- Association Rules
- Applying Association Rules on Big Data

Introduction

Introduction

- **Descriptive Analytics** use **Descriptive Data Mining** techniques which use **Unsupervised Machine Learning** techniques.

Descriptive Analytics → use → **Descriptive Data Mining** → use → **Unsupervised Machine Learning**

Clustering

Clustering

- **Clustering** is used to identify groups of similar objects in a multivariate data sets.
- As a data mining function, cluster analysis serves as a tool to gain insight into the distribution of data to observe characteristics of each cluster.
- Clustering analysis is broadly used in many applications in data mining:
  - Clustering can be used in **image processing**, for example, in a video k-means analysis can be used to **identify objects** in the video.
  - Clustering can help markets **characterize their customer groups** based on the purchasing patterns.
  - Clustering helps in **classifying documents** on the web for **information discovery**.
  - Clustering is used in **outlier detection** applications as detection of **credit card fraud**.

Clustering: Methods

- There are different **types of clustering** methods, including:
  - Partitioning clustering
  - Hierarchical clustering
  - Fuzzy clustering
  - Density-based clustering
  - Grid-based clustering
  - Model-based clustering
  - Constraint-based clustering

Clustering: Methods

- **Partitioning clustering:**
  - Suppose we are given a database of 'n' objects, the partitioning method constructs 'k' partition of data. Each partition will represent a cluster where each cluster contains at least one object, and each object must belong to exactly one cluster.
  - For a given number of partitions (say k), the partitioning method will create an initial partitioning.
  - Then it uses the iterative relocation technique to improve the partitioning by moving objects from one cluster to other.

Clustering: Methods

- **Hierarchical clustering:**
  - This method creates a hierarchical decomposition of the given set of data objects. There are two approaches here:
    - **Agglomerative Approach:** Known also as the bottom-up approach. In this, we start with each object forming a separate group. It keeps on merging the objects or groups that are close to one another until the termination condition holds.
    - **Divisive Approach:** Known also as the top-down approach. In this, we start with all of the objects in the same cluster. In the continuous iteration, a cluster is split up into smaller clusters until the termination condition holds

Clustering: Methods

- **Fuzzy clustering:**
  - Fuzzy clustering is also known as **soft** method. Standard clustering approaches produce partitions, in which each observation belongs to only one cluster. This is known as **hard** clustering.
  - In Fuzzy clustering, items can be a member of more than one cluster. Each item has a set of membership coefficients corresponding to the degree of being in a given cluster.
- **Density-based clustering:**
  - In density-based clustering, clusters are defined as areas of **higher density** than the remainder of the data set. Objects in sparse areas - that are required to separate clusters - are usually considered to be **noise** points.

Clustering: Methods

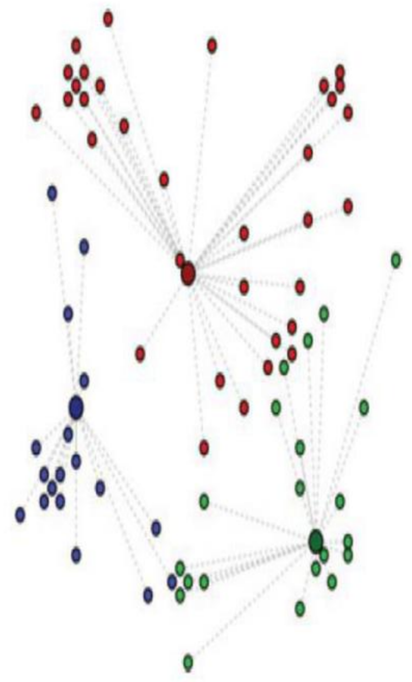
- **Grid-based clustering:**
  - In this method, the objects together form a **grid**. The object space is divided into **finite number of cells** that form a grid structure.
  - Cells are chosen randomly until all cells are traversed.
  - The density of a cell ' c' is calculated and if this density is greater than threshold density, then mark cell 'c' as a new cluster.
  - The densities of the neighboring cells are calculated and based on the threshold value it is decided to merge it with cell 'c'.

Clustering: Methods

- **Model-based clustering:**
  - **Model-based clustering** assumes that the data were generated by a **model** and tries to recover the original **model** from the data. The **model** that we recover from the data then defines **clusters** and an assignment of objects to **clusters**. A commonly used criterion for estimating the **model** parameters is maximum likelihood.
  - This method provides a way to automatically determine the number of clusters based on standard statistics, taking outlier or noise into account.
- **Constraint-based clustering:**
  - In this method, the clustering is performed by the incorporation of **user or application-oriented constraints.** A constraint refers to the user expectation or the properties of desired clustering results. Constraints provide us with an interactive way of communication with the clustering process.

Clustering: K-means

- Given a collection of objects each with n measurable attributes, ***k-means*** is an analytical technique that, for a chosen value of k, identifies k clusters of objects based on the objects' proximity to the center of the k groups.
- The center is determined as the arithmetic average (mean) of each cluster's n-dimensional vector of attributes.
- It is a Partitioning clustering method.



Possible k-means clusters for k=3

Clustering: K-means

- The k-means algorithm to find k clusters can be described in the following four steps:
  1. Choose the value of k and the k initial guesses for the centroids.
  2. Compute the distance from each data point to each centroid. Assign each point to the closest centroid. This defines the first k clusters.
     - For a given point, **pi** (pi1, pi2, ..pin) and a centroid, **q**, (q1, q2,..qn) where n is the number of features, the distance, **d**, between pi and q, is expressed as shown in the following equation:

$$d(p_i, q) = \sqrt{\sum_{j=1}^{n} (p_{ij} - q_j)^2}$$

Clustering: K-means

- The k-means algorithm to find k clusters can be described in the following four steps (cont.):
  3. Compute the centroid, the center of mass, of each newly defined cluster from Step 2. The centroid, ***q,*** of a cluster of **m** points, where each point pi has n features(pi1, pi2, ..pin), is calculated as in the following equation:

$$(q_1, q_2, \ldots, q_n) = \left(\frac{\sum_{i=1}^{m} p_{i1}}{m}, \frac{\sum_{i=1}^{m} p_{i2}}{m}, \ldots, \frac{\sum_{i=1}^{m} p_{in}}{m} \right)$$

  4. Repeat Steps 2 and 3 until the algorithm converges to an answer.
     - Convergence can be reached when the computed centroids do not change or the centroids and the assigned points oscillate back and forth from one iteration to the next.

Clustering: K-means

- **Determining the Number of Clusters (i.e. K):**
  - The value of k can be chosen based on a reasonable guess or some predefined requirement.
  - However, even then, it would be good to know how much better or worse having k clusters versus k – 1 or k + 1 clusters would be in explaining the structure of the data.
  - Next, a heuristic using the **Within Sum of Squares** (WSS) metric is examined to determine a reasonably **optimal value of k**.

Clustering: K-means

- **Determining the Number of Clusters (i.e. K):**
  - **WSS** is defined as shown in the following equation:

$$WSS = \sum_{i=1}^{M} d(p_i, q^{(i)})^2 = \sum_{i=1}^{M} \sum_{j=1}^{n} \left(p_{ij} - q_j^{(i)} \right)^2$$

  - **WSS** is the sum of the squares of the distances between each data point and the closest centroid. **M** is the number of objects in the dataset. The term $q^{(i)}$ indicates the closest centroid that is associated with the ith point. If the points are relatively close to their respective centroids, the WSS is relatively small. Thus, if k + 1 clusters do not greatly reduce the value of WSS from the case with only k clusters, there may be little benefit to adding another cluster.

Clustering: K-means

- **Advantages:**
  - Fast convergence for clustering small datasets
  - Easy to implement
- **Drawbacks:**
  - Computationally expensive for large datasets
  - Doesn't guarantee to converge to a global minimum. It is sensitive to the centroids' initialization. Different setups may lead to different results.
  - Strong sensitivity to outliers

Clustering: K-means++

- **K-means++**
  - K-means algorithm is sensitive to the initialization of the centroids.
  - **K-means++** is an algorithm for choosing the initial values for the K-means clustering algorithm to ensure a smarter initialization of the centroids
  - Apart from initialization, the rest of the algorithm is the same as the standard K-means algorithm.
  - First centroid is chosen randomly from the data points.
  - The next centroid is chosen from the data points such that **the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid**.
  - K-means++ is more likely to converge and run faster than K-means.

Clustering: k-medoids

- **K-Medoids** (also called as Partitioning Around Medoid - **PAM**):
  - It is known to be less sensitive to outliers than k-means.
  - The PAM algorithm searches for k representative objects in a data set (**k medoids**) and then assigns each object to the closest medoid in order to create clusters.
  - These medoids are actual observations from the dataset and not computed points (mean value) like in the case of k-means.
  - Its aim is to minimize the sum of dissimilarities between the objects in a cluster and the center of the same cluster (medoid).

Applying Clustering on Big Data

Applying Clustering on Big Data

- We now consider how to reformulate **K-means** algorithms for solving the clustering problem on **Big Data**.
- We will formulate this algorithm using **MapReduce** technique.
- Each iteration of standard k-means can be divided into two phases, the first of which **computes the sets of points closest to mean μi** , and the second of which **computes new means** of these sets.
- These two phases correspond to the **Map** and **Reduce** phases of the MapReduce algorithm.

Applying Clustering on Big Data

- The Map phase operates on each point x in its split.
- For a given x, find the mean μi (the cluster) which minimizes the **squared distance between x and the mean.**
- Then a <key-value> pair is emitted with this **mean's index i as key** and the **value x i.e <i, x>**
- Note that in order for the Map function to compute the distance between a point x and each of the means, each machine in our distributed cluster **must have the current set of means**. We must therefore **broadcast** the new means at each iteration.

Applying Clustering on Big Data

- The output of the map phase is huge, so a **combiner** is used to minimize the size of the data.
- The combiner calculates the average of the data instances for each cluster id, along with the number of the instances.
- It outputs **< i, (cluster mean, number of instances)>**

Applying Clustering on Big Data

- The **reduce phase** calculates the means by iterating over the values of the same cluster index.
- The shared means values have to be updated to the new computed values.
- **All the mentioned procedure is repeated until the convergence criterion is met.**

Association Rules

Association Rules

- Association rules method is an unsupervised learning method.
- This is a descriptive method often used to discover **interesting relationships hidden in a large dataset**.
- The disclosed relationships can be represented as rules or frequent itemsets.
- Here are some possible questions that association rules can answer:
  - Which products tend to be purchased together?
  - Of those customers who are similar to this person, what products do they tend to buy?
  - Of those customers who have purchased this product, what other similar products do they tend to view or purchase?

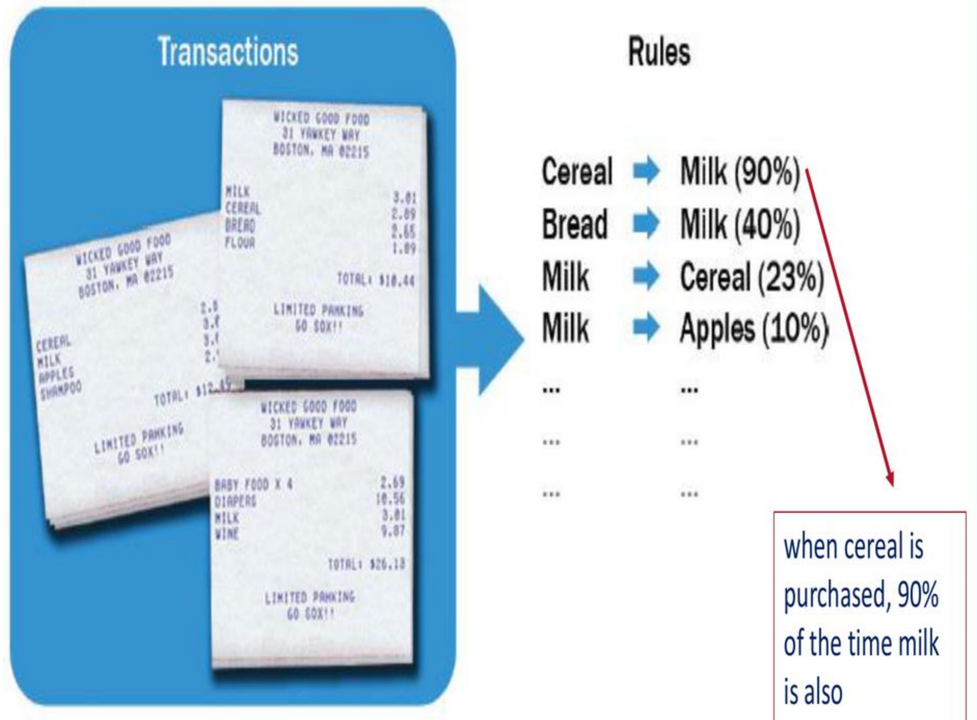
Association Rules: Overview

- For example, given a large collection of transactions in which each transaction consists of one or more items:

⇒association rules go through the items being purchased to see what items are frequently bought together and to discover a list of rules that describe the purchasing behavior.

- **The goal with association rules is to discover interesting relationships among the items.**
- The relationship occurs too frequently to be random and is meaningful from a business perspective.

Association Rules: Overview



The general logic behind association rules

Association Rules: Overview

- Each of the uncovered rules is in the form $X \rightarrow Y$, meaning that when item **X** is **observed**, item **Y** is **also observed**.
- In fact, because of their popularity in mining customer transactions, association rules are sometimes referred to as ***market basket analysis.***
- Each transaction can be viewed as the shopping basket of a customer that contains one or more items.
- This is also known as an ***itemset.***
- An itemset containing ***k*** items is called a ***k-itemset.***

Association Rules: Overview

- The term ***itemset*** generally refers to a collection of items or individual entities that contain some kind of relationship.
- This could be:
  - a set of retail items purchased together in one transaction,
  - a set of hyperlinks clicked on by one user in a single session,
  - a set of tasks done in one day,
  - others

Association Rules: Apriori Algorithm

- **Apriori** is one of the earliest and the most fundamental algorithms for generating association rules.
- It pioneered the use of support for pruning the itemsets and controlling the exponential growth of candidate itemsets.
- This approach eliminates the need for all possible itemsets to be enumerated within the algorithm, since the number of all possible itemsets can become exponentially large.

Association Rules: Apriori Algorithm

- One major component of Apriori is **support**.
- Given an itemset L, the **support** of L is the percentage of transactions that contain L.
- For example, if 80% of all transactions contain itemset {bread}, then the support of {bread} is 0.8.
- A ***frequent itemset*** has items that appear together often enough. The term "often enough" is formally defined with a ***minimum support*** criterion.
- If the minimum support is set at 0.5, any itemset can be considered a frequent itemset if at least 50% of the transactions contain this itemset.

Association Rules: Apriori Algorithm

- If an itemset is considered frequent, then any subset of the frequent itemset must also be frequent. If an itemset is infrequent, all its supersets will be infrequent.
- This is referred to as the ***Apriori property*** (or ***downward closure property***).
- For example, if 60% of the transactions contain {bread,jam}, then at least 60% of all the transactions will contain {bread} or {jam}.
- In other words, when the support of {bread,jam} is 0.6, the support of {bread} or {jam} is at least 0.6.
- **The Apriori property provides the basis for the Apriori algorithm.**

Association Rules: Apriori Algorithm steps

- **Step1: Generate 1-itemsets (L1)**
  - **Determine all the possible items** (or 1-itemsets, for example {bread}, {eggs}, {milk}, …)
  - **then identify which among them are frequent:**
    - Assuming the minimum support threshold (or the minimum support criterion) is set at 0.5:
    - **the algorithm identifies and retains those itemsets** that appear in at least 50% of all transactions
    - and **discards (or "prunes away") the itemsets** that have a support less than 0.5.

Association Rules: Apriori Algorithm steps

- **Step2: Generate 2-itemsets (L2)**
  - **the identified frequent 1-itemsets are paired into 2-itemsets** (for example, {bread,eggs}, {bread,milk}, {eggs,milk}, …)
  - and again evaluated to **identify the frequent 2-itemsets among them**.

Association Rules: Apriori Algorithm steps

- **Step3: Generate 3-itemsets (L3)**
  - Here we use **Apriori Property** for the generation of the candidate set of three itemsets. We perform two steps: **Join** and **Prune**
  - For the generation of 3-itemsets we compute **L2 join L2**. Condition of joining Lk-1 and Lk-1 is that it should have **(K-2) elements in common**.
  - After that, we move to the **Prune step** to reduce the size: Check if all subsets of these item sets are frequent or not (what triplets can we make that do not contain any pair that was eliminated?)
  - After that, **compute the support** of the remaining 3-itemsets and identify which among them are frequent and eliminate the others.

Association Rules: Apriori Algorithm steps

- The same procedure is repeated in the **next iterations** of the Apriori algorithm.
- **At each iteration**, the algorithm checks whether the support criterion can be met;
  - if it can, the algorithm grows the itemset, repeating the process until it runs out of support or until the itemsets reach a predefined length.

Association Rules: Apriori Algorithm steps

- To **generate the association rules**, find all the rules of the frequent itemset such that these rules have **higher confidence value than the threshold or minimum confidence**.
- Confidence is the percent of transactions that contain both X and Y out of all the transactions that contain X and computed as follows:

$$Confidence(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X)}$$

Association Rules: Apriori Algorithm example

- **Example:** Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:

TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

Association Rules: Apriori Algorithm example

- **Step-1**: Assume minimum support count is 2, Calculate the support count for each itemset and prune those items with support count less than the minimum support count.

Itemset	Support_Count
A	6
B	7
C	5
D	2
E	1

Itemset	Support_Count
A	6
B	7
C	5
D	2

Item E is pruned since it doesn't satisfy the minimum support count criteria

Association Rules: Apriori Algorithm example

- **Step-2**: Create pairs of the frequent itemsets that are identified in Step-1 and again calculate the support count for each itemset and prune those itemsets with support count less than the minimum support count.

Itemset	Support_Count
{A, B}	4
{A,C}	4
{A, D}	1
{B, C}	4
{B, D}	2
{C, D}	0

Itemset	Support_Count
{A, B}	4
{A, C}	4
{B, C}	4
{B, D}	2

Association Rules: Apriori Algorithm example

- **Step-3**: Expand the frequent itemsets that are identified in Step-2 to include 3 items and again calculate the support count for each itemset and prune those itemsets with support count less than the minimum support count.

Itemset	Support_Count
{A, B, C}	2
{A, B, D}	0
{B, C, D}	1

Itemset	Support_Count
{A, B, C}	2

Note that the supports of {A,B,D} and {B,C,D} are not computed since their 2-itemsets subsets are not frequent i.e. {A,D} and {C,D} then any superset is also not frequent. Apriori property is used here.

Association Rules: Apriori Algorithm example

- **Step-4**: Generating the association rules; we will now apply on the last frequent itemset reached (generally frequent itemsets from the other iterations are considered).

Rules	Support	Confidence
A ^B → C	2	Sup{(A ^B) ^C}/sup(A ^B)= 2/4=0.5=50%
B^C → A	2	Sup{(B^C) ^A}/sup(B ^C)= 2/4=0.5=50%
A^C → B	2	Sup{(A ^C) ^B}/sup(A ^C)= 2/4=0.5=50%
C→ A ^B	2	Sup{(C^(A ^B)}/sup(C)= 2/5=0.4=40%
A→ B^C	2	Sup{(A^(B ^C)}/sup(A)= 2/6=0.33=33.33%
B→ B^C	2	Sup{(B^(B ^C)}/sup(B)= 2/7=0.28=28%

Association Rules: Apriori Algorithm example

- **Step-4**: We will exclude the rules that have less confidence than the minimum threshold (50%).
- Therefore, the rules selected will be:
  - {A, B} → C
  - {B, C} → A
  - {A, C} → B

Association Rules: Evaluation of Candidate Rules

- We have seen two measures which are: **Support** and **Confidence.**
- There are other measures such as: **Lift** and **Leverage**
- ***Lift*** measures how many times more often X and Y occur together than expected if they are statistically independent of each other. Lift is a measure of how X and Y are really related rather than coincidentally happening together:

$$Lift(X \rightarrow Y) = \frac{Support(X \wedge Y)}{Support(X) * Support(Y)}$$

A larger value of lift (greater than 1) suggests a greater strength of the association between X and Y.

Association Rules: Evaluation of Candidate Rules

- ***Leverage*** is a similar notion, but instead of using a ratio, leverage uses the difference. Leverage measures the difference in the probability of X and Y appearing together in the dataset compared to what would be expected if X and Y were statistically independent of each other.

$$Leverage(X \rightarrow Y) = Support(X \wedge Y) - Support(X) * Support(Y)$$

A larger leverage value indicates a stronger relationship between X and Y.

Association Rules: Evaluation of Candidate Rules

- Confidence is able to identify **trustworthy rules**, but it cannot tell whether a **rule is coincidental**. **Why?**
- A high-confidence rule can sometimes be misleading because confidence **does not consider support** of the itemset in the **rule consequent**.
- Measures such as lift and leverage not only ensure interesting rules are identified but also **filter out the coincidental rules**.

Applying Association Rules on Big Data

Applying Association Rules on Big Data

- A MapReduce job splits the input transaction database into various blocks.
- The map task parses one transaction at a time and extracts each itemset included in the transaction it received as input.
- After processing, the mapper sends the itemset to the partitioner by emitting the itemset and frequency as <key, value> pair, where **'key 'is a candidate itemset and "value" is 1**.

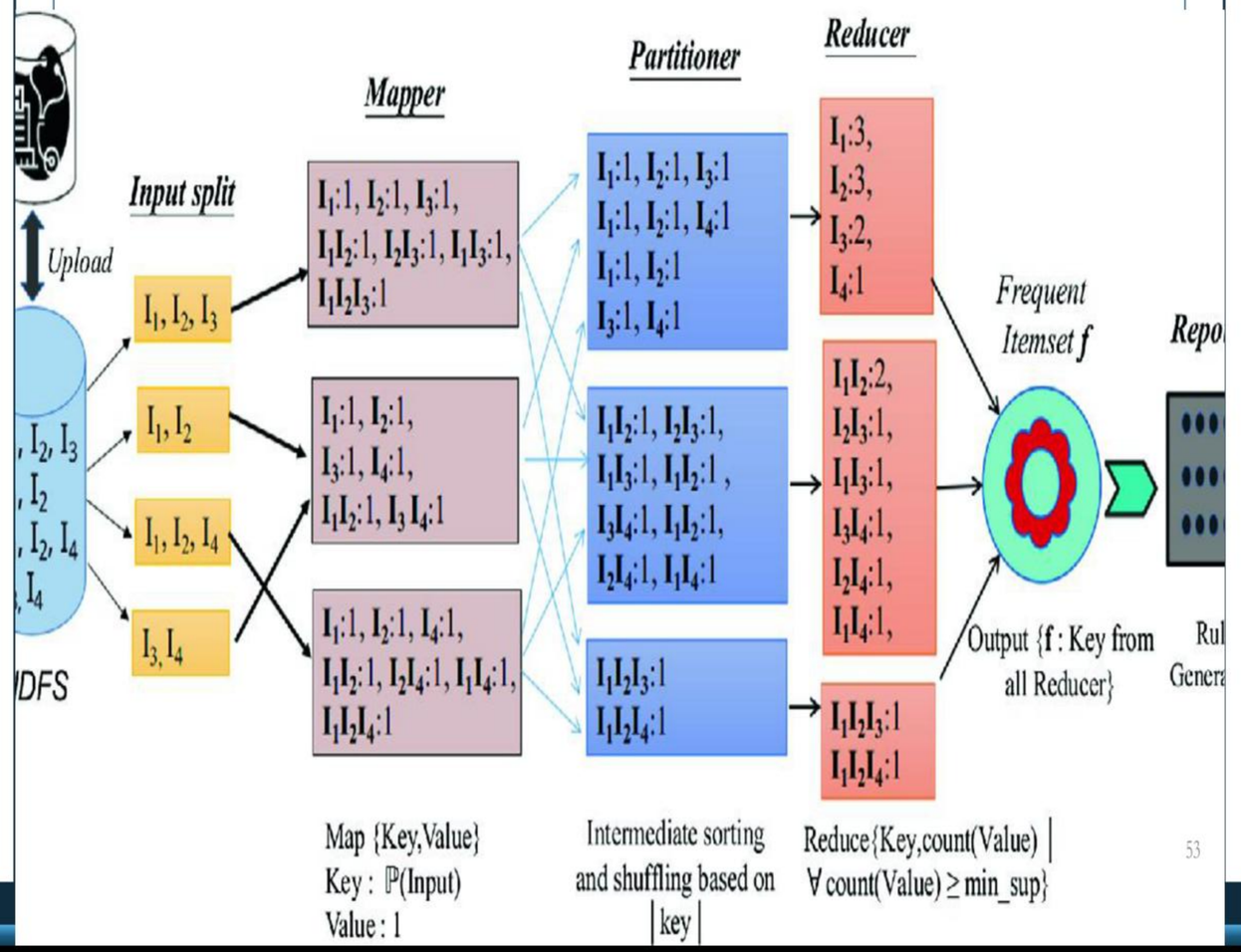
Applying Association Rules on Big Data

- The partition task collects all the intermediate <key, value> pair emitted from the map task as its input.
- Based on the **key size**, i.e., the size of each itemset, the partitioner specifies that all the values for each itemset are grouped together and maps all the values of a single key to go to the same reducer.
- The output of all partitioner are shuffled and exchanged to make the list of values associated with the same key as <key,list(value)> pairs.

Applying Association Rules on Big Data

- The reduce task collects each key passing all the values emitted against the same key as arguments, i.e., <key,list(value)> pairs emitted by partitioner task.
- Then, it **sums up** the values of respective keys. Candidate itemset whose sum of values is **supportcount<minimum_support_count** is discarded.
- The result from all reducers is written to the output file.

Applying Association Rules on Big Data



References

- Bodoia, M. (2016). MapReduce Algorithms for k-means Clustering. Stanford University
- Bhattacharya N., Mondal S., Khatua S. (2019) A MapReduce-Based Association Rule Mining Using Hadoop Cluster—An Application of Disease Analysis. In: Saini H., Sayal R., Govardhan A., Buyya R. (eds) Innovations in Computer Science and Engineering. Lecture Notes in Networks and Systems, vol 74.