ALY6110

Data Management and Big Data

Instructor: Valeriy Shevchenko

Northeastern

Lecture 2 (Week 2)

Big Data Projects Methodology

Learning Objectives

This session will explain:

- New Paradigm for Big Data
- Big Data Architecture and Models
- Recent Trends in Technology
- Big Data Projects Methodology

New Paradigm for Big Data

- In the past decade the amount of data being created has skyrocketed.
- More than 30,000 gigabytes of data are generated every second, and the rate of data creation is only accelerating.
- The data we deal with is diverse:
  - Blog Posts
  - Tweets
  - Social Network Interactions
  - Photos
  - Audio Video
  - Log messages from servers
  - Scientific measurements
  - .....
- This astonishing growth in data has profoundly affected businesses.

From Traditional Data to Big Data

- Traditional database systems, such as relational databases, have been pushed to the limit.
- In an increasing number of cases these systems are breaking under the pressures of “Big Data.”
- Traditional systems, and the data management techniques associated with them, are failing to scale to Big Data.
- To tackle the challenges of Big Data, a new breed of technologies has emerged.
- Many of these new technologies have been grouped under the term *NoSQL*.
- In some ways, these new technologies are more complex than traditional databases, and in other ways they’re simpler.

From Traditional Data to Big Data

- These systems can scale to vastly larger sets of data, but using these technologies effectively requires a fundamentally new set of techniques.
- They aren't one-size-fits-all solutions.
- Many of these Big Data systems were pioneered by Google:
  - Distributed filesystems.
  - MapReduce computation framework.
  - Distributed locking services.
- Another notable pioneer in the space was Amazon
  - Innovative distributed key/value store called Dynamo.
- The open source community responded in the years following with:
  - Hadoop
  - Hbase
  - MongoDB
  - Cassandra
  - RabbitMQ
  - ...

Traditional Data Problems

- How the traditional database technology hits the limits?

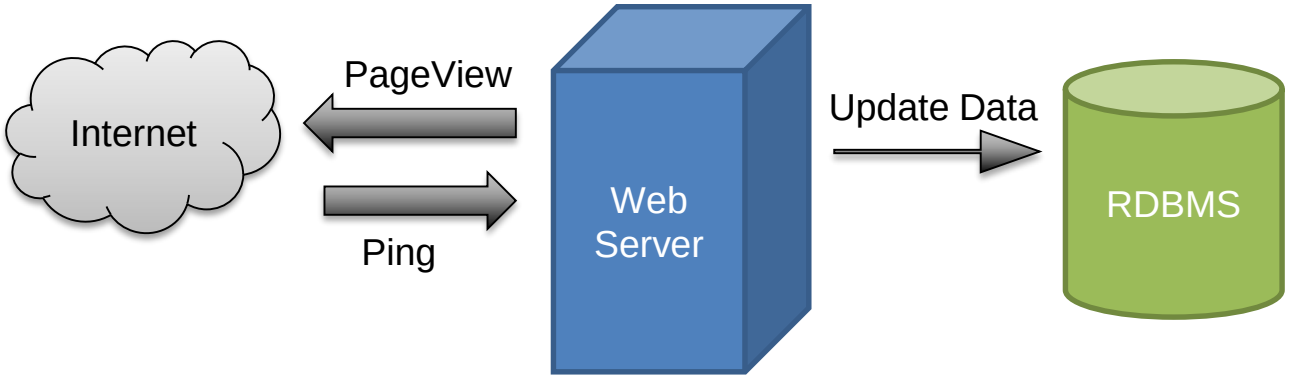
- Simple project :
  - Build simple web analytics application
  - Application should track the number of pageviews for any URL customer wants to track
  - Customers web page pings the application web server every time pageview is received
  - Application should tell at any point what the top 100 URLs are by the number of pages.

- Traditional relational schema would look like this:

Column Name	Type
ID	Integer
User_ID	Integer
url	varchar(1000)
pageview	bigint

Traditional Data Problems

- Your back end consists of an RDBMS with a table of that schema and a web server.
- Whenever someone loads a web page being tracked by your application, the web page pings your web server with the pageview,
- Web server increments the corresponding row in the database.

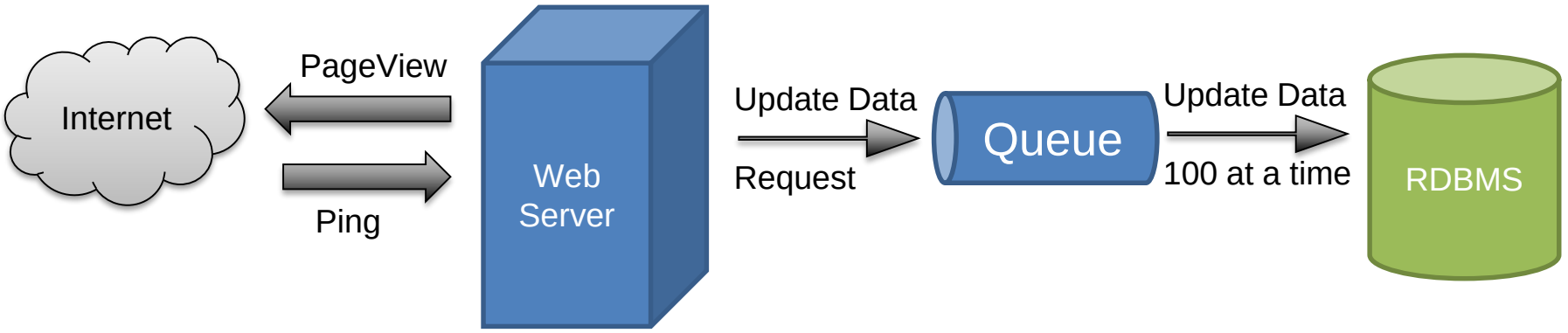


Traditional Data Problems

- The web analytics product is a huge success, and traffic to your application is growing.
- You start getting lots of emails from your monitoring system: “Timeout error on inserting to the database.”
- Problem is obvious: the database can’t keep up with the load, so write requests to increment pageviews are timing out.
- You realize that it’s wasteful to only perform a single increment at a time to the database.
- The DB updates be more efficient if you batch many increments in a single request.
- So you re-architect your back end to make this possible.

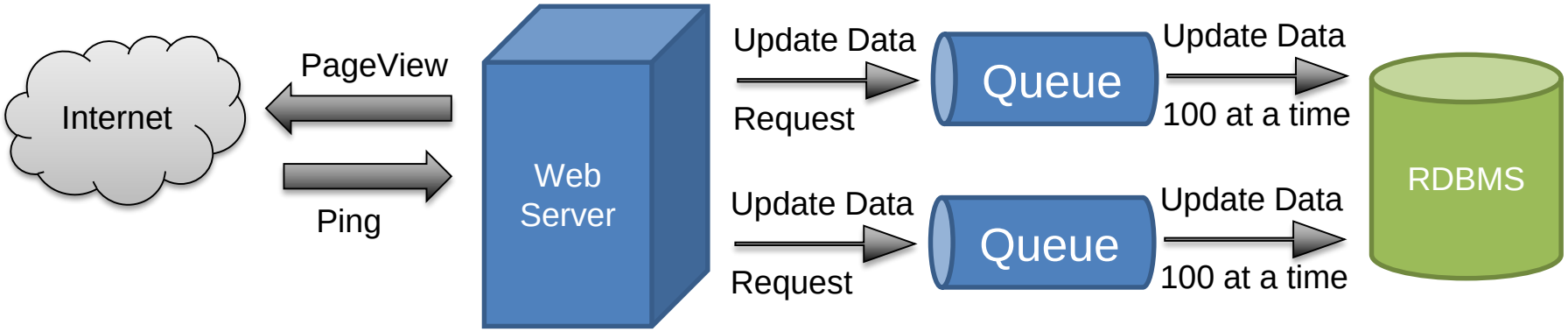
Traditional Data Problems

- Instead of having the web server hit the database directly, you insert a queue between the web server and the database.
- Whenever you receive a new pageview, that event is added to the queue.
- You then create a worker process that reads 100 events at a time off the queue, and batches them into a single database update.



Traditional Data Problems

- Adding a queue and doing batch updates was only a temporary patch for the scaling problem.
- The application continues to get more and more popular, and again the database gets overloaded.
- The worker process can't keep up with the writes, so you try adding more workers to parallelize the updates.
- Unfortunately that doesn't help; the database is clearly the bottleneck.



Traditional DB Problems

- You do some Google searches for how to scale a write-heavy relational database.
- You find that the best approach is to use multiple database servers and spread the table across all the servers.
- Each server will have a subset of the data for the table.
- This is known as *horizontal partitioning* or *sharding*.
- This technique spreads the write load across multiple machines.
- The sharding technique you use is to choose the shard for each key by taking the hash of the key modded by the number of shards.

Traditional DB Problems

- You do some Google searches for how to scale a write-heavy relational database.
- You find that the best approach is to use multiple database servers and spread the table across all the servers.
- Each server will have a subset of the data for the table.
- This is known as *horizontal partitioning.*
- This technique spreads the write load across multiple machines.
- You write a script to map over all the rows in your single database instance, and split the data into four shards (partitions).

Traditional DB Problems

- Your application code needs to know how to find the shard for each key.
- Your wrap a library around your database-handling code that reads the number of shards from a configuration file, and you redeploy all of your application code.
- You have to modify your top-100-URLs query to get the top 100 URLs from each shard and merge those together for the global top 100 URLs.
- As the application gets more and more popular, you keep having to “reshard” the database into more shards to keep up with the write load.

Traditional DB Problems

- Each time gets more and more painful because there's so much more work to coordinate.
- You have to do all the "resharding" in parallel and manage many active worker scripts at once.
- You forget to update the application code with the new number of shards, and it causes many of the increments to be written to the wrong shards.
- So you have to write a one-off script to manually go through the data and move whatever was misplaced.

Traditional DB Problems

- Eventually you have so many shards that it becomes a not-infrequent occurrence for the disk on one of the database machines to go bad.
- That portion of the data is unavailable while that machine is down.
- You update your queue/worker system to put increments for unavailable shards on a separate “pending” queue that you attempt to flush once every five minutes.
- You use the database’s replication capabilities to add a slave to each shard so you have a backup in case the master goes down.

Traditional DB Problems

- While working on the queue/worker code, you accidentally deploy a bug to production that increments the number of pageviews by two, instead of by one, for every URL.
- You don’t notice until 24 hours later, but by then the damage is done.
- Your weekly backups don’t help because there’s no way of knowing which data got corrupted.
- After all this work trying to make your system scalable and tolerant of machine failures, your system has no resilience to a human making a mistake.

Traditional DB Problems

- As the simple web analytics application evolved, the system continued to get more and more complex: queues, shards, replicas, resharding scripts, and so on.
- Developing applications on the data requires a lot more than just knowing the database schema.
- Your code needs to know how to talk to the right shards, and if you make a mistake, there's nothing preventing you from reading from or writing to the wrong shard.

Traditional DB – What Wen Wrong?

- The traditional database is not self-aware of its distributed nature, so it can't help you deal with shards, replication, and distributed queries.
- All that complexity got pushed to you both in operating the database and developing the application code.
- The system is not engineered for human mistakes.
- The system keeps getting more and more complex, making it more and more likely that a mistake will be made.

Big Data Techniques

- The Big Data techniques address the scalability and complexity issues in a dramatic fashion.
- The databases and computation systems you use for Big Data are aware of their distributed nature by design.
- Instead of storing the pageview counts as your core dataset, which you continuously mutate as new pageviews come in, you store the raw pageview information.
- That raw pageview information is never modified.
- When you make a mistake, you might write bad data, but at least you won't destroy good data.

Big Data Techniques

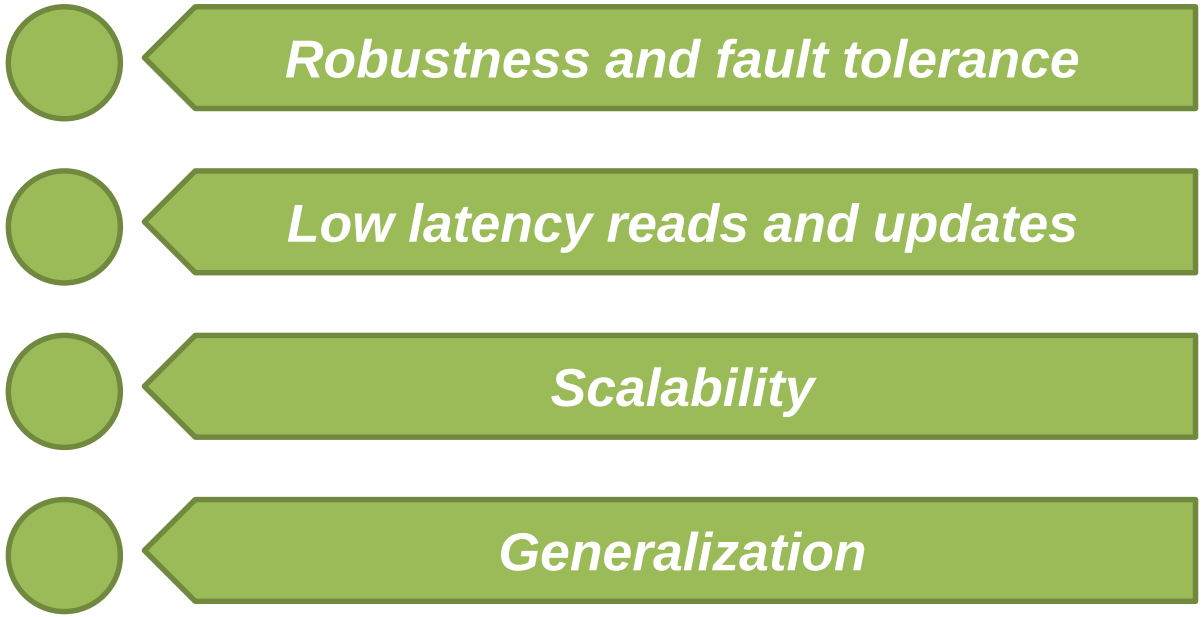
- Large-scale computation systems like Hadoop and databases such as Cassandra and Riak address Big Data demands and requirements
- These systems can handle very large amounts of data, but with serious trade-offs.
- Hadoop, for example, can parallelize large-scale batch computations on very large amounts of data, but the computations have high latency. You don't use Hadoop for anything where you need low-latency results.
- NoSQL databases like Cassandra achieve their scalability by offering you a much more limited data model than you're used to with something like SQL.

Big Data Architecture

- These tools on their own are not a panacea.
- When intelligently used in conjunction with one another, you can produce scalable systems for arbitrary data problems with human-fault tolerance and a minimum of complexity.
- This called the Lambda Architecture
- The properties we expect from Big Data systems are as much about complexity as they are about scalability.
- Not only must a Big Data system perform well and be resource efficient, it must be easy to reason about as well.

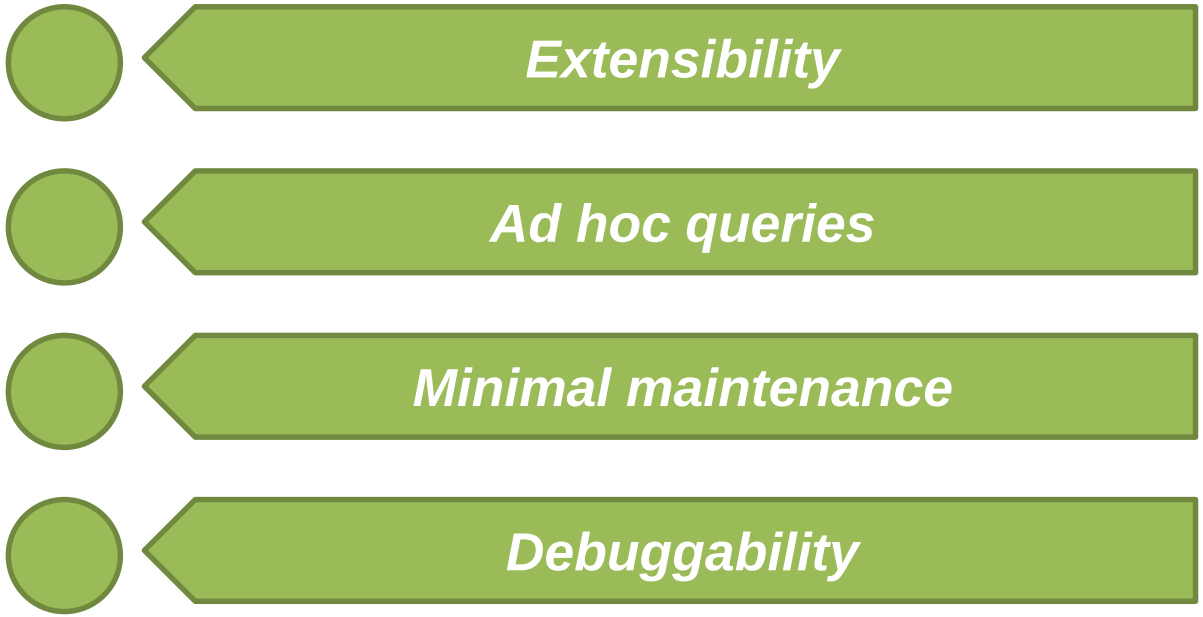
Big Data Architecture (cont'd)

- Desired properties of a Big Data system:



Big Data Architecture (cont'd)

- Desired properties of a Big Data system:



Big Data Architecture (cont'd)

- Systems need to behave correctly despite:
  - Machines going down randomly
  - The complex semantics of consistency in distributed databases
  - Duplicated data
  - Concurrency
- Avoid complexities
- Systems has to be resilient to human error by providing a clear and simple mechanism for recovery

Big Data Architecture (cont'd)

- The vast majority of applications require reads to be satisfied with very low latency
- The update latency requirements vary a great deal between applications.
- Some applications require updates to propagate immediately, but in other applications a latency of a few hours is fine.
- Low latency reads and updates must be achieved without compromising the robustness of the system.

Big Data Architecture (cont'd)

- Scalability is the ability to maintain performance in the face of increasing data or load by adding resources to the system.
- The Lambda Architecture is horizontally scalable across all layers of the system stack: scaling is accomplished by adding more machines.

Big Data Architecture (cont'd)

- A general system can support a wide range of applications.

- Because the Lambda Architecture is based on functions of all data, it generalizes to all applications:
  - Financial management systems
  - Social media analytics
  - Scientific applications
  - Social networking
  - ...

Big Data Architecture (cont'd)

- Extensible systems allow functionality to be added with a minimal development cost.
- Part of making a system extensible is making it easy to do large-scale migrations.
- Being able to do big migrations quickly and easily is core to the approach of system extensibility

Big Data Architecture (cont'd)

- Being able to do ad hoc queries on your data is extremely important.
- Nearly every large dataset has unanticipated value within it.
- Being able to mine a dataset arbitrarily gives opportunities for business optimization and new applications.

Big Data Architecture (cont'd)

- Maintenance is the work required to keep a system running smoothly:
  - Add machines to scale
  - Keep processes up and running
  - Debugging anything that goes wrong in production
- Combat implementation complexity by relying on simple algorithms and simple components.
- The more complex a system, the more likely something will go wrong
- The Lambda Architecture pushes complexity out of the core components and into pieces of the system whose outputs are discardable after a few hours.

Big Data Architecture (cont'd)

- A Big Data system must provide the information necessary to debug the system when things go wrong.
- The key is to be able to trace, for each value in the system, exactly what caused it to have that value.
- "Debuggability" is accomplished in the Lambda Architecture through the functional nature of the batch layer and by preferring to use recomputation algorithms when possible.

Lambda Architecture

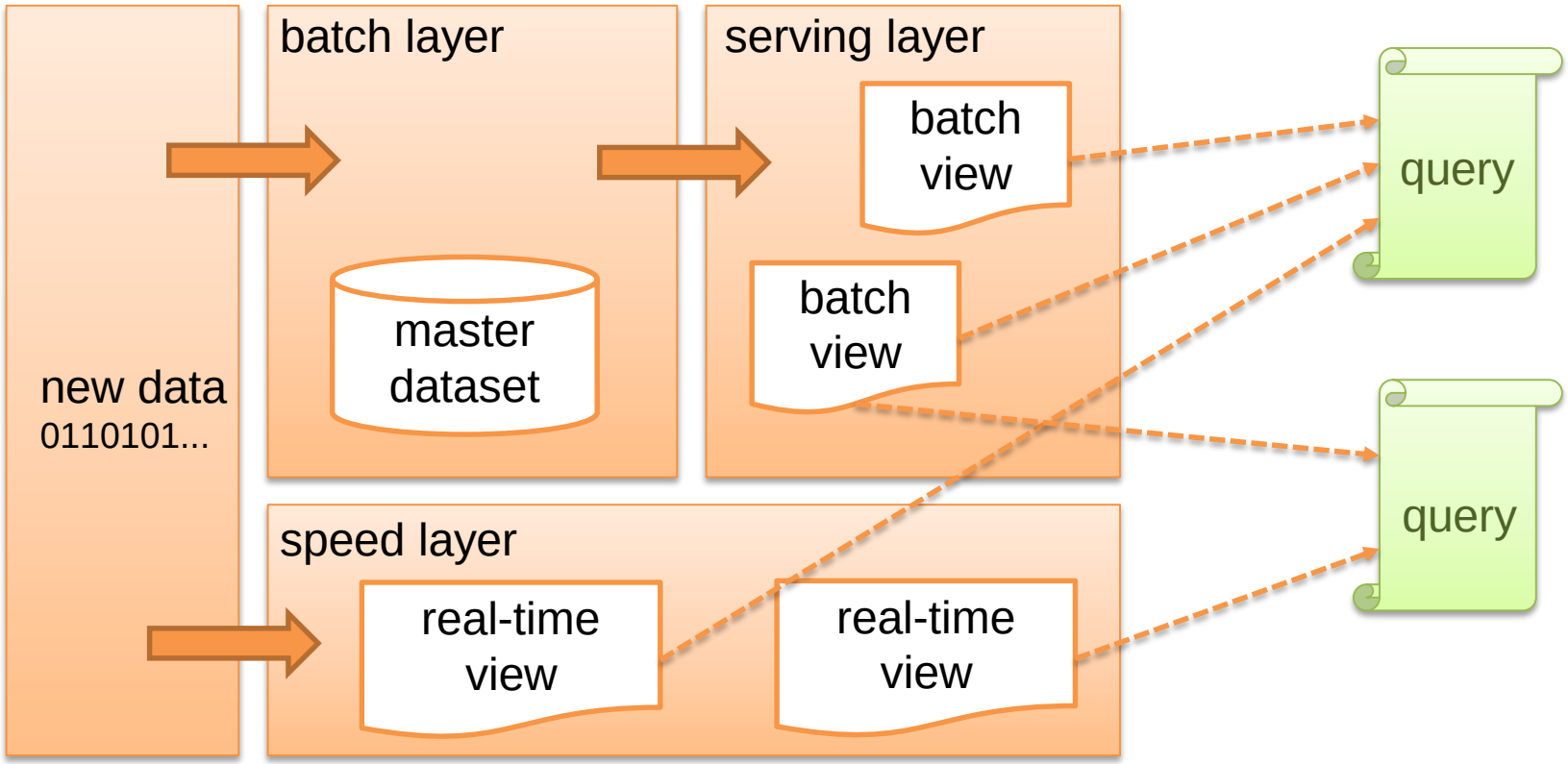
- Nathan Marz came up with the term Lambda Architecture (LA) for a generic, scalable and fault-tolerant data processing architecture, based on his experience working on distributed data processing systems at Backtype and Twitter.

- The Lambda Architecture aims to satisfy the needs for a robust system that is fault-tolerant, both against hardware failures and human mistakes, being able to serve a wide range of workloads and use cases, and in which low-latency reads and updates are required.

- The resulting system should be linearly scalable, and it should scale out rather than up.

Lambda Architecture (cont'd)

- The main idea of the Lambda Architecture is to build Big Data systems as a series of layers
- Lambda Architecture Overview:



Lambda Architecture (cont'd)

1. All **data** entering the system is dispatched to both the batch layer and the speed layer for processing.

2. The **batch layer** has two functions:
   - managing the master dataset (an immutable, append-only set of raw data)
   - pre-compute the batch views.

3. The **serving layer** indexes the batch views so that they can be queried in low-latency, ad-hoc way.

4. The **speed layer** compensates for the high latency of updates to the serving layer and deals with recent data only.

5. Any incoming **query** can be answered by merging results from batch views and real-time views.

Lambda Architecture (cont'd)

- The portion of the Lambda Architecture that implements the *batch view = function(all data)* equation is called the ***batch layer.***
- The batch layer stores the master copy of the dataset and precomputes batch views on that *master dataset*.
- The batch layer needs to be able to do two things:
  - store an immutable, constantly growing master dataset.
  - compute arbitrary functions on that dataset.
- The batch layer runs in a while(true) loop and continuously recomputes the batch views from scratch.
- Hadoop is the canonical example of a batch-processing system.

Lambda Architecture (cont'd)

- Serving layer is a specialized distributed database that loads in a batch view and makes it possible to do random reads on it.
- When new batch views are available, the serving layer automatically swaps those in so that more up-to-date results are available.
- A serving layer database supports batch updates and random reads.
- It doesn't need to support random writes which makes it extremely simple.
- ElephantDB, the serving layer database is only a few thousand lines of code.

Lambda Architecture (cont'd)

- Speed layer computes arbitrary functions on arbitrary data in real time to compensate for those last few hours of serving layer updates
- Its goal is to ensure new data is represented in query functions as quickly as needed for the application requirements
- Speed layer only looks at most recent data, whereas the batch layer looks at all the data at once
- In order to achieve the smallest latencies possible, the speed layer doesn't look at all the new data at once.
- It updates the real-time views as it receives new data instead of recomputing the views from scratch like the batch layer does.

Recent Trends in Technology

CPUs aren't getting faster

- We've started to hit the physical limits of how fast a single CPU can go
- To scale to more data we have to parallelize computations.
- We see the rise of shared-nothing parallel algorithms and their corresponding systems, such as MapReduce.
- Instead of scale by buying a better machine (*vertical scaling*), systems scale by adding more machines (*horizontal scaling*).

Elastic clouds

- We see the rise of elastic clouds, also known as *Infrastructure as a Service (Amazon Web Services)*.
- Elastic clouds let you increase or decrease the size of your cluster nearly instantaneously, so if you have a big job you want to run, you can allocate the hardware temporarily.
- Elastic clouds dramatically simplify system administration.

Recent Trends in Technology (cont'd)

Vibrant open source ecosystem for Big Data

- *Batch computation systems* - Hadoop project:
  - Hadoop Distributed File System (HDFS) - distributed, fault-tolerant storage system that can scale to petabytes of data
  - Hadoop MapReduce - horizontally scalable computation framework that integrates with HDFS.
- *Serialization frameworks* - provide tools and libraries for using objects between languages. They can serialize an object into a byte array from any language, and then deserialize that byte array into an object in any language.
- *Random-access NoSQL databases* - Cassandra, HBase, MongoDB, Voldemort, Riak, CouchDB, and others. They sacrifice the full expressiveness of SQL and instead specialize in certain kinds of operations.
- *Messaging/queuing systems* - provide a way to send and consume messages between processes in a fault-tolerant and asynchronous manner. Message queue is a key component for doing real-time processing.(*Apache Kafka*)
- *Real-time computation systems* - high throughput, low latency, stream-processing systems.(*Storm*)

We've Got the Data, What now?

- Depending on the type of data scale, different data quality and preprocessing techniques can be used.
- The quality of the models, charts and studies in data analytics depends on the quality of the data being used.
- The nature of the application domain, human error, the integration of different data sets and the methodology used to collect data can generate data sets that are noisy, inconsistent, or contain duplicate records.
- Even with a large number of robust descriptive and predictive algorithms available to deal with noisy, incomplete, inconsistent or redundant data, an increasing number of real applications have their findings harmed by poor-quality data.

Data Quality

- In data sets collected directly from storage systems (actual data), it is estimated that noise can represent 5% or more of the total data set
- When these data are used by algorithms that learn from data (ML algorithms) - the analysis problem can look more complex than it really is if there is no data pre-processing.
- This increases the time required for the induction of assumptions or models and resulting in models that do not capture the true patterns present in the data set.
- The elimination or even just the reduction of these problems can lead to an improvement in the quality of knowledge extracted by data analysis processes.

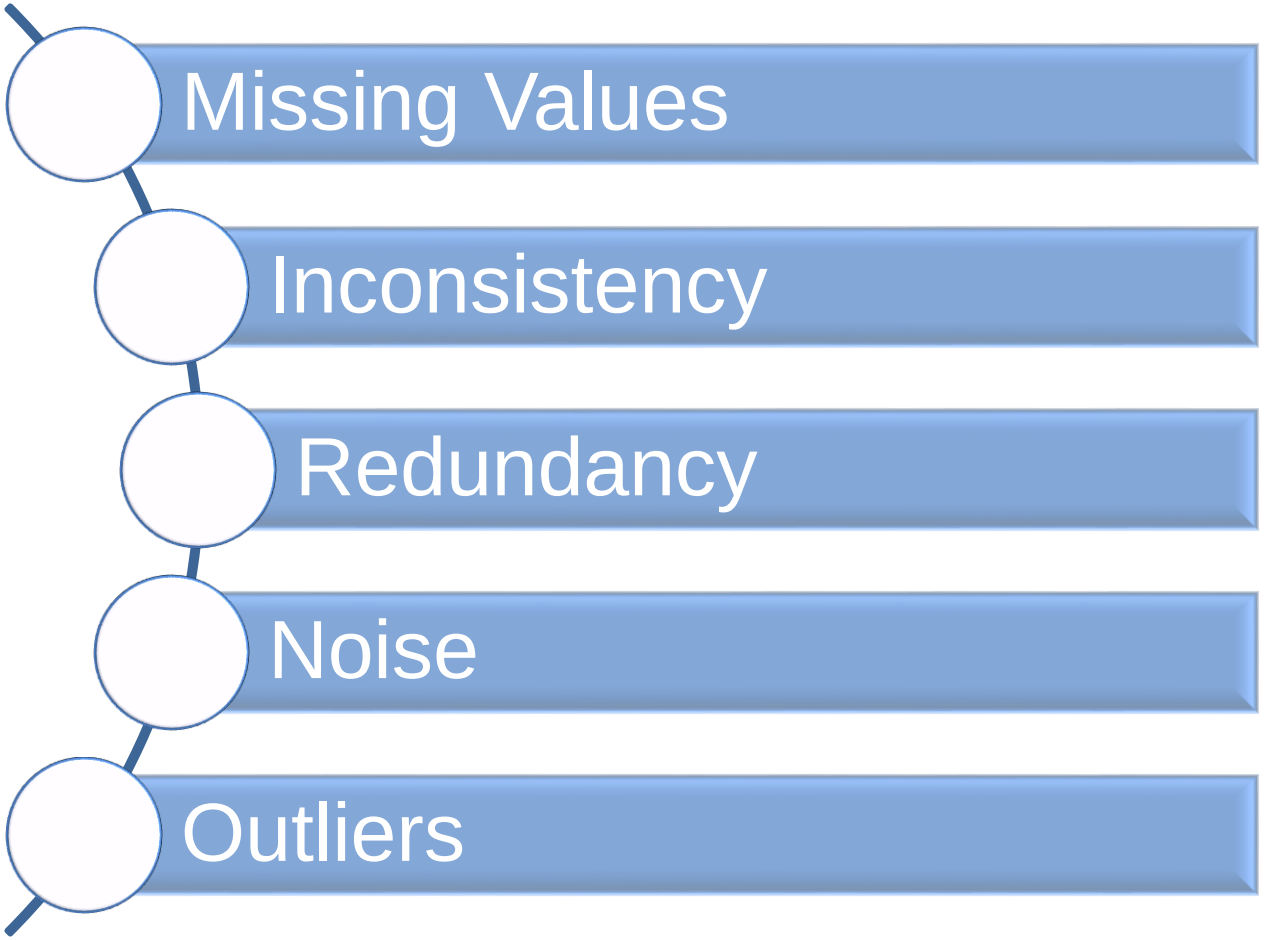
Data Quality

- Data quality is important and can be affected by internal and external factors:

External factors are related to faults in the data collection process, and can involve the absence of values for some attributes and the voluntary or involuntary addition of errors to others.

Internal factors can be linked to the measurement process and the collection of information through the attributes chosen.

Data Quality Main Problems



Data Quality Main Problems (cont'd)

- In real-life applications, it is common that some of predictive attribute values for some of the records may be missing in the data set.
- There are several causes of missing values, among them:
  - attributes values only recorded for some time after start of collection.
  - the value of an attribute being unknown at time of collection.
  - distraction, misunderstanding or refusal at time of collection.
  - attribute not required for particular objects.
  - non-existence of a value.
  - fault in the data collection device.
  - cost or difficulty of assigning a class label to an object in classification problems.
- Since many data analysis techniques were not designed to deal with a data set with missing values, the data set must be pre-processed.

Data Quality Main Problems (cont'd)

Several approaches can be used:

- *Ignore missing values*
  - Use for each object only the attributes with values, without paying attention to missing values.
  - Modify a learning algorithm to allow it to accept and work with missing values.
- *Remove objects*
  - Use only those objects with values for all attributes.
- *Make estimates*
  - Fill the missing values with estimates based on values for this attribute in the other objects.

Data Quality Main Problems (cont'd)

- A data set can also have inconsistent values.
- The presence of inconsistent values in a data set usually reduces the quality of the model induced by ML algorithms.
- Inconsistent values can be found in the predictive and/or target attributes.
- An example of an inconsistent value in a predictive attribute is a zip code that does not match the city name. This inconsistency can be due to a mistake or a fraud.
- A good policy to deal with inconsistent values in the predictive attribute is to treat them as missing values.

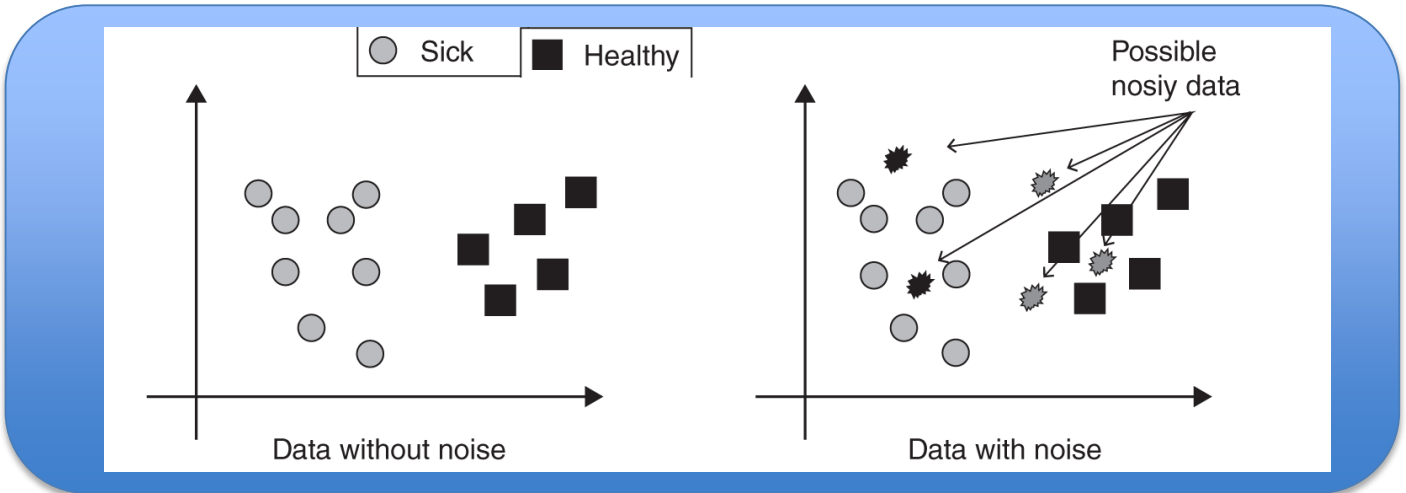
Data Quality Main Problems (cont'd)

- Redundant objects are those that do not bring any new information to a data set.
- Objects very similar to other objects represent an irrelevant data
- Redundancy occurs mainly in the whole set of attributes.
- Redundant data could be due to small mistakes or noise in the data collection (*same addresses for people whose names differ by just a single letter*).
- Redundant data can be duplicate data. Deduplication is a preprocessing technique whose goal is to identify and remove copies of objects in a data set

Data Quality Main Problems (cont'd)

Noise

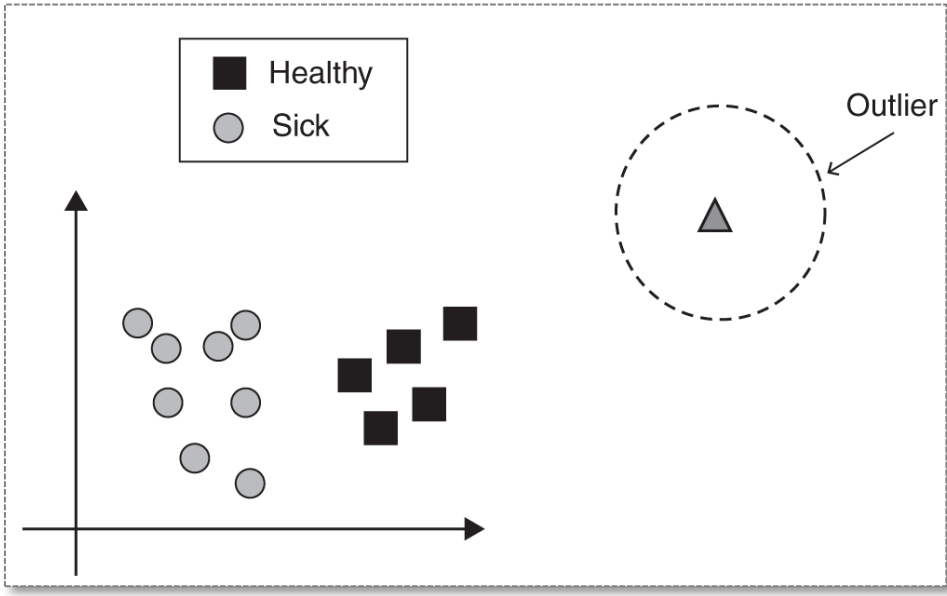
- Noisy data are data that do not meet the set of standards expected for them.
- Noise can be caused by incorrect or distorted measurements, human error or even contamination of the samples.
- Noise detection can be performed by adaptation of classification algorithms or by the use of noise filters for data preprocessing.



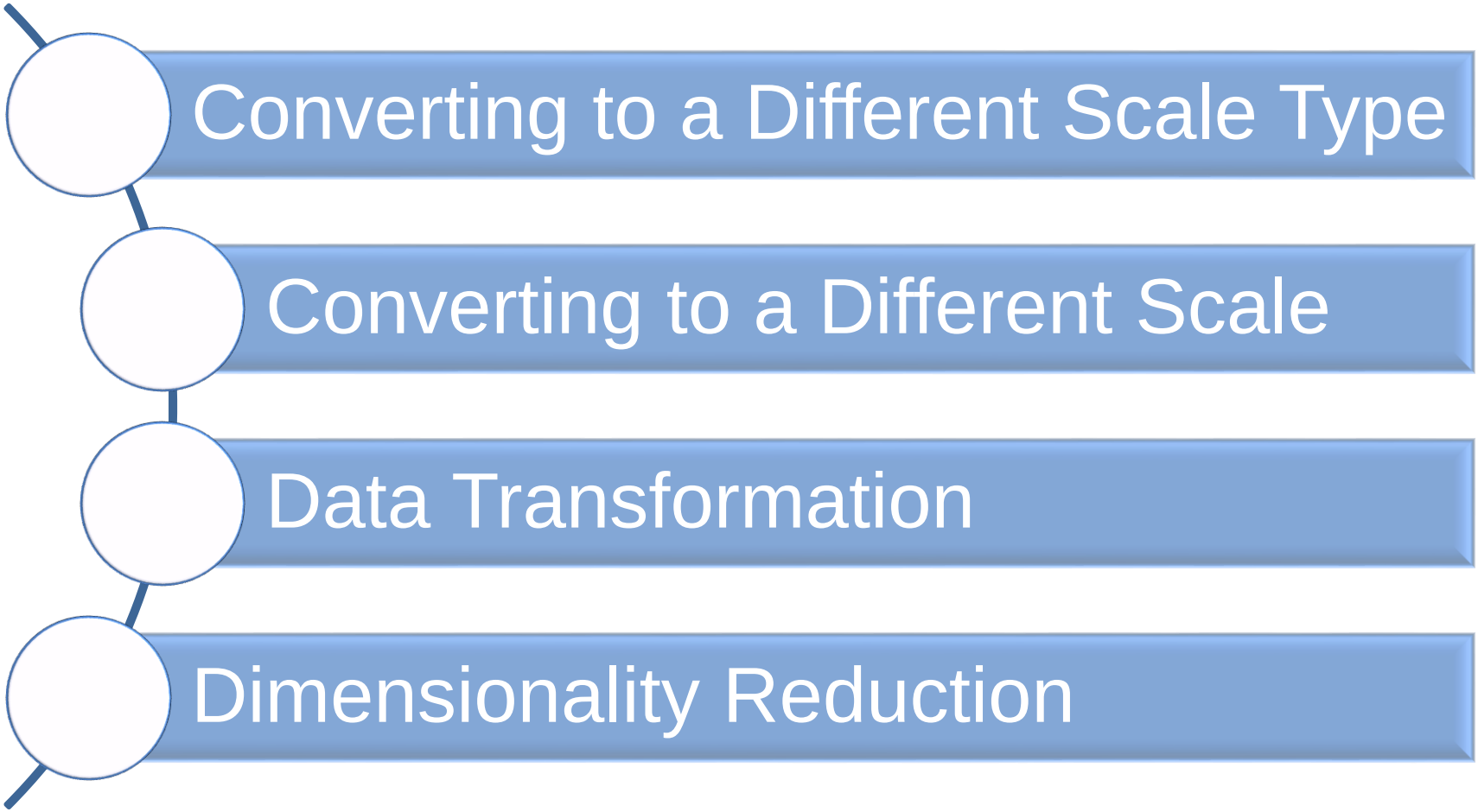
Data Quality Main Problems (cont'd)

Outliers

- In a data set, outliers are anomalous values or objects.
- They can also be defined as objects whose values for one or more predictive attributes are very different from the values found in the same predictive attributes of other objects.
- In contrast to noisy data points, outliers can be legitimate values.
- There are several data analysis applications whose main goal is to find outliers in a data set.
- Particularly in anomaly detection tasks, the presence of outliers can indicate the presence of noise.



Data Pre-Processing



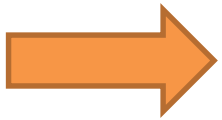
Data Pre-Processing (cont'd)

Converting to a Different Scale Type – Nominal to Relative

- Some ML algorithms can use only data of a particular scale type.
- Data can be converted from a qualitative scale to a quantitative one.

Original Data

Food	Age	Distance	Company
Chinese	51	Close	Good
Italian	43	Very close	Good
Italian	82	Close	Good
Burgers	23	Far	Bad
Chinese	46	Very far	Good
Chinese	29	Too far	Bad
Burgers	42	Very far	Good
Chinese	38	Close	Bad
Italian	31	Far	Good



Converted Data

F1	F2	F3	Age	Distance	Company
0	0	1	51	Close	Good
0	1	0	43	Very close	Good
0	1	0	82	Close	Good
1	0	0	23	Far	Bad
0	0	1	46	Very far	Good
0	0	1	29	Too far	Bad
1	0	0	42	Very far	Good
0	0	1	38	Close	Bad
0	1	0	31	Far	Good

Data Pre-Processing (cont'd)

Converting to a Different Scale Type – Ordinal to Relative or Absolute

- For ordinal values, the conversion is more intuitive, since we can convert to natural numbers, starting with the value 0 for the smallest value and, for each subsequent value, adding 1 to the previous value.

Nominal	Natural Number	Gary Code	Thermometer Code
Small	0	00	000
Medium	1	01	001
Large	2	10	011
Huge	3	11	111

- Gray code keeps the distance between two consecutive values as a different value in one of the binary values.
- The thermometer code, starts with a binary vector with only 0 values and substitutes one 0 value by 1, from right to left, as the ordinal value increases.

Data Pre-Processing (cont'd)

Converting to a Different Scale Type – Relative or Absolute to Ordinal or Nominal

- Discretization is a process to convert quantitative values to nominal or ordinal ones.
- Discretization used to reduce the number of quantitative values.

Quantative values	Conversion by width	Conversion by frequency
2	A	A
3	A	A
5	A	A
7	A	B
10	B	B
15	B	B
16	C	C
19	C	C
20	C	C

- Discretization has two steps:
  - Define the number of quantative values - bins.
  - Define the interval of values to associate with each bin
- There are ate least two ways to define how to associate quantative values to a different bins:
  - Association by width
  - Association by frequency

Data Pre-Processing (cont'd)

Converting to a Different Scale Type – Relative or Absolute to Ordinal or Nominal

- Association by width - the intervals will have the same range: the same difference between the largest and smallest values:
  - 2-8, 8-15, 16-22 (the difference is 6)

Quantative values	Conversion by width	Conversion by frequency
2	A	A
3	A	A
5	A	A
7	A	B
10	B	B
15	B	B
16	C	C
19	C	C
20	C	C

- Association by frequency - each interval will have the same number of values:
  - 2-5, 7-15, 16-20

- Quantitative scale does not necessarily have to have all possible values.
- Upper and lower limits of the interval do not need to be in the data set and that some values, which will never appear, can be left out of the intervals.

Data Pre-Processing (cont'd)

Converting to a Different Scale

- Converting data in a scale to another scale of the same type is necessary in several situations, such as when using distance measures.
- This kind of conversion is typically done in order to have different attributes expressed on the same scale; a process known as "normalization".
- The normalization is carried out for each attribute individually.
- There are two ways to normalize the data:
  - Standardization.
  - Min–max rescaling.

Data Pre-Processing (cont'd)

Converting to a Different Scale – min-max rescaling

Friend Name	Age in Years	Education GPA	Rescaled Age (*amplitude* => *43-38=5*)	Rescaled Education (*amplitude* => *4.2-2.0 = 2.2*)
Bernhard	**43**	**2.0**	(**43**-38) / 5 = 1.0	(**2.0**-2.0) / 2.2 = 0.0
Gwyneth	**38**	**4.2**	(**38**-38) / 5 = 0.0	(**4.2**-2.0) / 2.2 = 1.0
James	**42**	**4.0**	(**42**-38) / 5 = 0.8	(**4.0**-2.0) / 2.2 = 0.91

- Min–max rescaling, converts numerical values to values in a given interval.

- To convert a set of values to values in the interval **[0.0, ..., 1.0]**, you simply subtract the smallest value from each of the values in the set and divide the new value by the amplitude: the difference between the maximum and minimum values.

Data Pre-Processing (cont'd)

Converting to a Different Scale – standardization

Friend Name	Age in Years	Education GPA	Standardized Age (Average=41 Deviation=2.65)	Standardized Education (Average=3.4 Deviation=1.22)
Bernhard	43	2.0	(43-41) / 2.65 = 0.75	(2.0-3.4) / 1.22 = -1.15
Gwyneth	38	4.2	(38-41) / 2.65 = -1.13	(4.2-3.4) / 1.22 = 0.66
James	42	4.0	(42-41) / 2.65 = 0.38	(4.0-3.4) / 1.22 = 0.48

- When using standardization, first subtract the average of the attribute values and then divide the result by the standard deviation of these values.
- As a result, the standardized values of the attribute will have now an average of 0.0 and standard deviation of 1.0.

Data Pre-Processing (cont'd)

Converting to a Different Scale – standardization

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2}$$

Friend Name	Age in Years
Bernhard	43
Gwyneth	38
James	42

- Calculating standard deviation:

1. (43 + 38 + 42) / 3 = 41 – **Mean**
2. $(43\text{-}41)^2 + (38\text{-}41)^2 + (42\text{-}41)^2 = 14$
3. 14 / (3 – 1)= 7 – **Variance**
4. $\sqrt{7} = 2.65$ – **Standard Deviation**

Data Pre-Processing (cont'd)

Data Transformation

- Another important issue for data summarization is transformations that might be necessary to perform to simplify the analysis or to allow the use of particular modeling techniques.

- Applying a logarithmic function to the values of a predictive attribute:

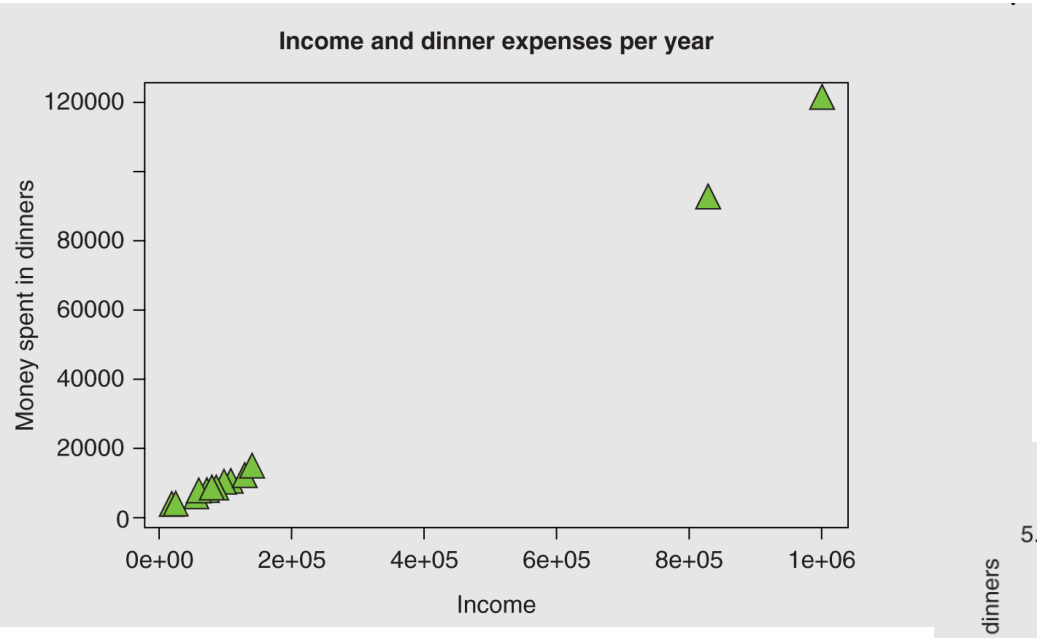
  usually performed for skewed distributions, when some of the values are much larger (or much smaller) than the others. he logarithm makes the distribution less skewed. Thus, log transformations make the interpretation of highly skewed data easier.

- Conversion to absolute values:

  For some predictive attributes, the value's magnitude is more important than its sign, if the value is positive or negative.

Data Pre-Processing (cont'd)

Data Transformation – Applying Logarithmic function example



Data Pre-Processing (cont'd)

Dimensionality Reduction

- The dimensionality reduction of a data set can bring several benefits:
  - Reduces the training time, decreases memory needed and improves performance of ML algorithms.
  - Eliminates irrelevant attributes and reduces the number of noisy attributes.
  - Allows the induction of simpler and more easily interpretable models.
  - Makes the data visualization easier to understand and allows visualization of data sets with a high number of attributes.
  - Reduces the cost of feature extraction, making ML-based technologies accessible to a larger number of people.
- There are two alternatives to reduce the number of attributes: attribute aggregation or attribute selection

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Aggregation

- In attribute aggregation, we replace a group of attributes with a new attribute: a combination of the attributes in the group.
- Attribute aggregation, also known as multidimensional scaling, reduces the data to a given number of attributes, allowing an easier visualization.
- Attribute aggregation techniques project the original data set into a new, lower-dimensional space, but keeping the relevant information.

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Aggregation

- Several techniques have been proposed for attribute aggregation in the literature.
  - Principal Component Analysis – PCA
  - Independent Component Analysis – ICA
  - Multidimensional Scaling – MDS
- Some of the techniques work by linearly combining the original attributes, creating a smaller number of attributes, referred to as a set of components.
- Other techniques can also create non-linear combinations of the original attributes of a data set.

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Aggregation

- A key concept to understand aggregation techniques is the concept of data projection.
- A projection transforms a set of attributes in one space to a set of attributes in another space.
- The original data are named sources and the projected data signals.
- A simple projection would be to remove some attributes from the original data set, creating a new data, the projected data set, with the remaining attributes.
- In a projection, we want to keep as much of the information present in the original set of attributes as possible.
- Ideally, the projection removes redundancy and noise from the original data.

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Selection

- Instead of aggregating attributes, another approach to reduce dimensionality is by selecting a subset of the attributes.
- This approach can speed up the learning process, since a smaller number of operations will need to be made.
- Attribute selection techniques can be roughly divided into three categories:
  - Filters.
  - Wrappers.
  - Embedded.

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Selection

Filters

- Filters look for relations between the predictive attribute values and the target attribute.
- Filters rank the attributes according to this relation.
- If the values of a predictive attribute have a strong relation with a label attribute value, this predictive attribute receives a high ranking position.

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Selection

Wrappers

- Wrappers explicitly use a classifier to guide the attribute selection process.
- Wrappers select the set of predictive attributes that provides the highest predictive performance for the classifier, regardless of the attribute ranking positions.
- Instead of ranking the predictive attributes, wrappers usually look for the subset of attributes with the best predictive ability.
- Wrappers capture the relationship between attributes and reducing the chance of selecting redundant attributes.

Data Pre-Processing (cont'd)

Dimensionality Reduction - Attribute Selection

Embedded

- In the embedded category, attribute selection is performed as an internal procedure of a predictive algorithm.
- One predictive technique able to perform embedded attribute selection is a decision tree induction algorithm.
- When these algorithms are applied to a data set, they produce a predictive model and select a subset of predictive attributes chosen be the most relevant for classification model induction.

Summary

In this session we've learned:

- The needs to handle Big Data
- Big Data Lambda Architecture
- Trends in Big Data technology
- Big Data Quality Problems
- Big Data Pre-Processing Techniques

References

www.wikipedia.org
lambda-architecture.net
www.mathisfun.com

Nathan Marz, James Warren
Big Data
Principles and Best Practices of Scalable Real-Time Data Systems

ALY6110

Data Management and Big Data

Q & A

Instructor: Valeriy Shevchenko
v.shevchenko@northeastern.edu