

Big Data Notes

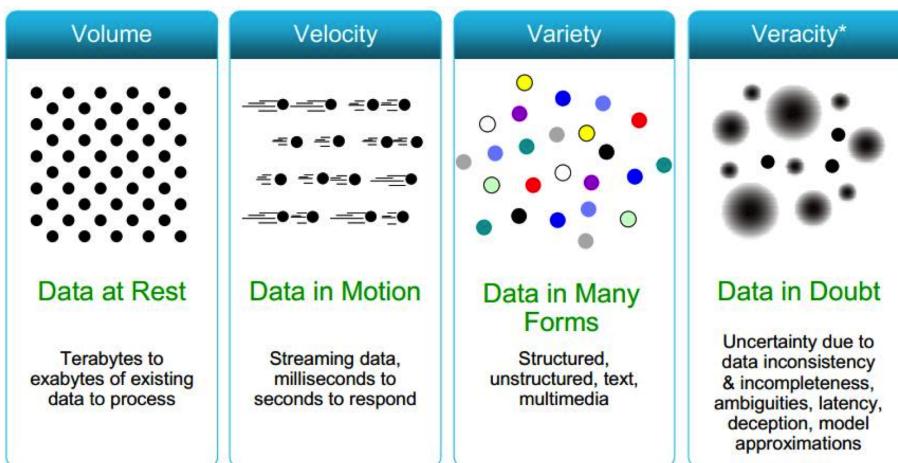
November 1, 2017

1 What is Big Data?

Big Data is data whose scale (volume), diversity, and complexity require new architecture, techniques, algorithms, and analytics to manage it and extract value and hidden knowledge from it.

Big Data presents several challenges:

- **Volume:** refers to the vast amount of data generated every second.
- **Velocity:** refers to the speed at which new data is generated and the speed at which data moves around.
- **Variety:** refers to the different types of data (structured, raw logs, semi-structured, etc.).
- **Veracity:** refers to the uncertainty or trustworthiness of the data.



The challenges presented by the 4 V's in Big Data revolution could not be overcome by traditional relational databases. Moreover, parallel and distributed database systems faced challenges in achieving linear scalability and dealing with the Variety and Veracity issues faced when processing Big Data.

In 2003/2004 Google published a paper on Google File System(GFS) and MapReduce which started the craze!! Google was facing issues in scalability due to the ever growing raw data generated by Web 2.0, social media (facebook, twitter, etc.), logs, user profiles (searches, click, transaction patterns, etc.). Specifically, Google wanted a scalable, self-managing, distributed processing system to regularly compute PageRank of the crawled web. Their goal was to build such an infrastructure on a large collection of inexpensive commodity hardware connected by Ethernet cables or inexpensive switches to perform processing of raw data....i.e. move away from expensive bulky compute servers and distributed databases.

Google published two papers based on their internal infrastructure:

- **Distributed File System (DFS):** larger file units that provide replication of data or redundancy to protect against the frequent media failures that occur when data is distributed over

thousands of low-cost compute nodes.

Ghemawat, S., Gobioff, H. Leung S. The Google File System. 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October, 2003.

- **MapReduce** : framework that allows users to run applications on computing clusters efficiently and in a way that is tolerant of hardware failures during the computation.
Dean, J. and Ghemawat, S. MapReduce: simplified data processing on large clusters. Communication of ACM 51, 1 (Jan. 2008), 107-113.

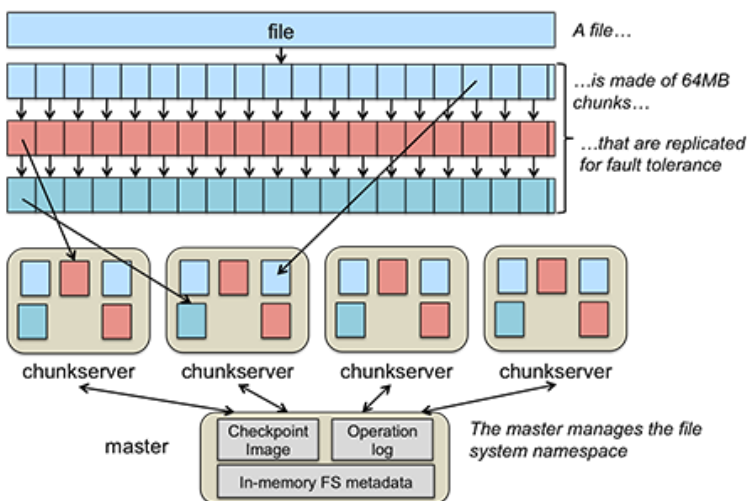
1.1 Google Distributed File System (GFS)

The Google Distributed File System (GFS) is a file system for large units of data that provide replication and redundancy to protect against the frequent media failures that occur when data is distributed over thousands of low-cost compute nodes.

A large file is split into chunks (64MB blocks) which is distributed across ‘**chunk servers**’. The chunks are replicated 2-3x for fault tolerance in case any chunk server goes down. The location of a given file’s chunks is stored at the ‘**master node**’ which maintains the meta data about the file and the chunk servers and manages the chunk servers. Client library contacts the master node to find the relevant chunk servers that contain the various chunks of a given file. Once the client knows all the relevant chunk servers, it will then contact the chunk servers directly to access the data.

Summary:

- Chunk servers
 - A large file is split into contiguous chunks (typically 64MB)
 - Each chunk is replicated (usually 2x or 3x) and stored on chunk server
 - Replication policy makes sure chunk servers are on two different racks.. in case of rack failures.
- Master node
 - Stores metadata about where files are stored



Once Google’s GFS paper was published, Yahoo wanted to utilize a similar file systems architecture for storing its large files. Hence, Yahoo started working on ‘Hadoop Distributed File System (HDFS)’ and made the project open source. HDFS is basically modeled after GFS with a few variations on the design, replication policy, etc. Here a mapping of the terminology between GFS and HDFS:

- Chunk servers are called ‘Data Node’ in HDFS
- Master node is called the ‘Name Node’ in HDFS

1.2 MapReduce

MapReduce is a programming model Google has used successfully to process its big-data sets (20000 peta bytes per day). Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, and handles machine failures, efficient communications, and performance issues.

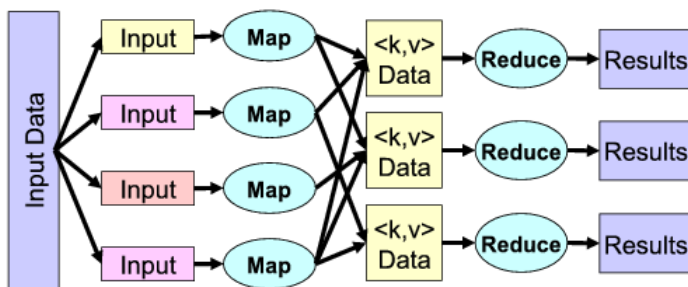
MapReduce is designed for one-off processing tasks, such as Extract, Transform and Load(ETL) and read-once applications. It is not designed for real-time processing or transaction-based applications. MapReduce provides an abstraction that allows engineers to perform simple computations while hiding the details of :

- Automatic parallelization of jobs
- I/O scheduling
- Data distribution Load balancing Fault tolerance

The canonical use of MR is characterized by the following template of five operations:

- Read logs of information from several different sources;
- Parse and clean the log data;
- Perform complex transformations on the data;
- Decide what attribute data to store;
- Load the information into a DBMS or other storage engine; and
- Complex analysis on various data from different sources

Google's MapReduce infrastructure was also made into a open-source platform with Yahoo's efforts. Yahoo called this platform 'Hadoop'. Hadoop's initial design was based on Google's MapReduce paper but many improvements and policy changes have been applied to this platform since the Google paper was published.



MapReduce contacts HDFS to locate the input file's relevant chunks and determines the 'input-splits' (essentially the records) that should be processed. Each Mapper job receives a set of input-splits which it processes to generate a key-value pair. Once all Mappers complete processing, the key-value pairs are then sorted and grouped by key. Each Reducer will receive all values for a given key, which it then will aggregate to generate a result.

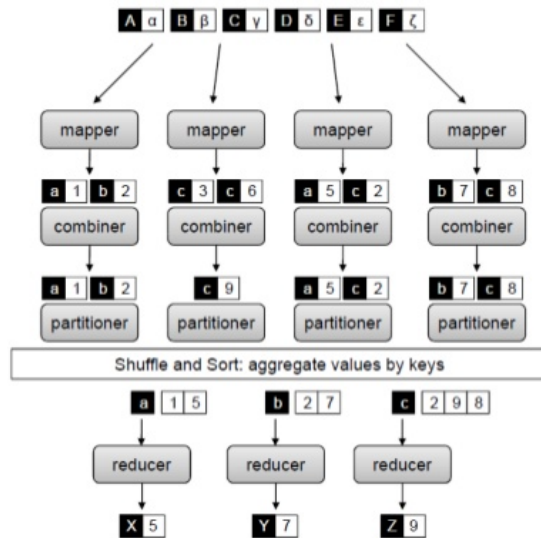
The MapReduce flow can also include a 'Combiner' and 'Partitioner' for optimization. The Combiner class is used in between the Map class and the Reduce class to reduce the volume of data transfer between Map and Reduce. Usually, the output of the map task is large and the data transferred to the reduce task is high.

Combiners are an optimization in MapReduce that allow for local aggregation before the shuffle and sort phase. When the map operation outputs its pairs they are already available in memory. When a combiner is used then the map key-value pairs are not immediately written to the output, but

instead grouped by key. Hence, doing a reduce-type function, but the operation must be associative and commutative.

The Partitioner class is used in between the Map class and the Reduce class (after the Combiner... if a combiner is used). Partitioners are responsible for dividing up the intermediate space and assigning intermediate key-value pairs to Reducers. Partitioners are used to ensure that approximately the same number of keys are assigned to each reducer (dependent on the quality of the hash function).

MapReduce with Partitioner and Combiner



On top of the MapReduce/Hadoop frameworks two tools were proposed to speed-up ETL processing and support ad-hoc queries and analysis on the data. While MapReduce offers the infrastructure for processing, querying, and analyzing large data, its complex and time-consuming to write such jobs. Thus, two tools were proposed (Apache Hive and Pig) for manipulating, querying, and analyzing large data sets. These two tools sit on-top of the Hadoop/MapReduce stack offer support for ETL operations and ad-hoc queries.

Apache Hive was proposed by Facebook as data-warehousing tools used mainly for data exploration and report generation. Hive offers a SQL-like language called HiveQL which offers a sub-set of the SQL syntax to query large tables. Hive allows users to defined table and 'load' files stored in HDFS into Hive tables; HiveQL can then be applied to query the tables or even perform joins between tables. HiveQL queries are converted into MapReduce jobs so that the programmer can work at a higher level than they would normally when writing MapReduce jobs.

Apache Pig was proposed by Yahoo as a ETL tool for designed specifically for data transformation and complex flow expression. Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs (this language is called Pig Latin). The programs written in Pig Latin are compiled into MapReduce jobs that have been optimized by the compiler.

Question: What is Hive? Hive can be thought of as a data warehouse infrastructure for providing summarization, query and analysis of data that is managed by Hadoop. Hive provides a SQL interface for data that is stored in Hadoop. And, it implicitly converts queries into MapReduce jobs so that the programmer can work at a higher level than he or she would when writing MapReduce jobs in Java. Hive is an integral part of the Hadoop ecosystem that was initially developed at Facebook and is now an active Apache open source project.

Question: What are the limitations of Hive? 1) The SQL syntax that Hive supports is quite restrictive. So for example, we are not allowed to do sub-queries, which is very very common in the SQL world. There is no windowed aggregates, and also ANSI joins are not allowed. 2) The other restriction that is quite limiting is the data types that are supported, for example when it comes to Varchar support or Decimal support, Hive lacks quite severely.

Question: What is the difference between Pig and Hive? You can compare the two by their use cases and/or limitations. Pig and Hive are fairly similar in their objective, which is to provide a higher level of abstraction over low-level MapReduce jobs. Developers often use basic functions such as aggregation, joins, etc and hence instead of describing these operations at a low-level and re-generating basic code these two platforms have presented solutions to ease with application development. Apache Hive offers a SQL-like query language that makes its adoption quite easy for those familiar with SQL. On the other-hand, Apache Pig offers much more flexibility and support for more complex multi-stage data processing applications. Pig is often used when data is in various formats (specifically semi-structured) and requires much more in-depth processing and analysis, while Hive is used by analysis to generate reports and mostly operates on ‘structured’ data with a defined schema. In either case, its best to utilize these tools rather than write your own MapReduce jobs when possible because these tools have optimized basic operators such as joins and aggregation.

2 Link-Analysis via MapReduce

Early search engines crawled the Web and extracted terms to build an inverted index, which maps a term to all relevant Web pages. When a search query is issued, the query is decomposed into search terms, and the pages with those terms are look-up from the inverted index and ranked based on importance (using TF-IDF, or cosine similarity function, etc.)

In 1996, it became content similarity (term-frequency) alone was no longer sufficient because of several factors:

- Number of pages grew rapidly in the mid-late 1990s
- A search query can result in 10M relevant pages, how to rank results suitably?
- Content similarity can be easily spammed by repeating term and add many relevant terms to boost ranking of the page, where in-fact the page may not be related to the search query.

During 1997-1998 two most influential link-based page ranking approaches were proposed. Both proposed algorithms exploit the link-structure of the Web to rank pages according to authority or importance:

1. **HITS:** Jon Kleinberg (Cornel University), at Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, January 1998.
2. **PageRank:** Sergey Brin and Larry Page, PhD students from Stanford University, at Seventh International World Wide Web Conference (WWW7) in April, 1998.

Google realized that with the ever growing inter-connectivity of the web, it can be modeled as a directed graph. A link from one page to another is a directed edge in the graph. With this model, we can make the assumption that nodes with higher number of incoming links, are more likely to be important. Thus, Google decided to rank the pages by the link structure. Page Rank (PR) was used to simulate where a Web surfer, starting at a random page, could congregate by following links from one page to another. PR is essentially a vote by all other pages on the Web about the importance of another page.

- A page is more important if it has more in-coming links (those are the ones the page cannot control).
- Are all incoming links equal? Nah links from more important pages count more

2.1 Page Rank Formulation

Suppose that a page P_j has L_j links. If one of those links is to page P_i , then P_j will pass on $\frac{1}{L_j}$ of its importance to P_i .

The importance ranking of P_i is then the sum of all the contributions made by the pages linking to it. That is, if we denote the set of pages linking to P_i by B_i , then :

$$PR(P_i) = \sum_{P_j \in B_i} \frac{PR(P_j)}{L_j}$$

The PR is usually computed iteratively (vs algebraically). The transition matrix M (n by n matrix, where n is the number of pages) represents the probability distribution of the location of a random surfer step j . The vector V represents the principle eigenvector of M .

The probability x_i that a random surfer will be at node i at the next step:

$$x_i = \sum M_{ij} V_j$$

V_j is the probability that the surfer was at node j at the previous step

M_{ij} is the probability that a surfer at node j will move to node i at the next step.

If we surf any of the n pages of the Web with equal probability

1. The initial vector v_0 will have $\frac{1}{n}$ for each component
2. After one step, the distribution of the surfer will be Mv_0
3. After two steps, $M(Mv_0) = M^2v_0$ and so on
4. Multiplying the initial vector v_0 by M a total of i times gives the distribution of the surfer after the i^{th} steps

The distribution of the surfer approaches a limiting distribution v that satisfies $v = Mv$ provided two conditions are met:

- The graph is strongly connected : It is possible to get from any node to any other node
- There are no dead ends : Dead ends = nodes that have no outgoing links

To avoid dealing with dead ends and non-strongly connected graphs, we modify the calculation of PageRank to allow each random surfer a small probability of teleporting to a random page rather than following an out-link from their current page. The iterative step, where we compute a new vector estimate of PageRanks v' from the current PageRank estimate v and the transition matrix M is

$$v' = \beta Mv + \frac{e(1-\beta)}{n}$$

Where,

β is a chosen constant (usually in the range 0.8 to 0.85)

e is a vector with 1s for the appropriate number of components

n is the number of nodes in the Web graph

The term βMv represents the case where the random surfer decides to follow an out-link from their present page (with probability β). The term $(1-\beta)/n$ is a vector representing the introduction, with probability $1-$, of a new random surfer at a random page.

Question 1: Consider that you are calculating PageRank values for web pages. There are 10 Billion web pages and you have created a 10 Billion x 10 Billion transition matrix M . As a part of iterative computations, you use the MapReduce computing framework without Taxation. The k^{th} iteration of the MapReduce job will create a vector v^k with 10 Billion items. The j^{th} item in v^k is calculated using the following formula:

$$v_j^{(k+1)} = \sum_j m_{ij} v_j^k$$

What are the values m_{ij} stored in the transition matrix M ?

- A: The total number of times that web page i has been visited
- B: The probability that page i is to be visited from the j^{th} page
- C: The page i 's page rank value after j^{th} iteration
- D: Random number generated by server

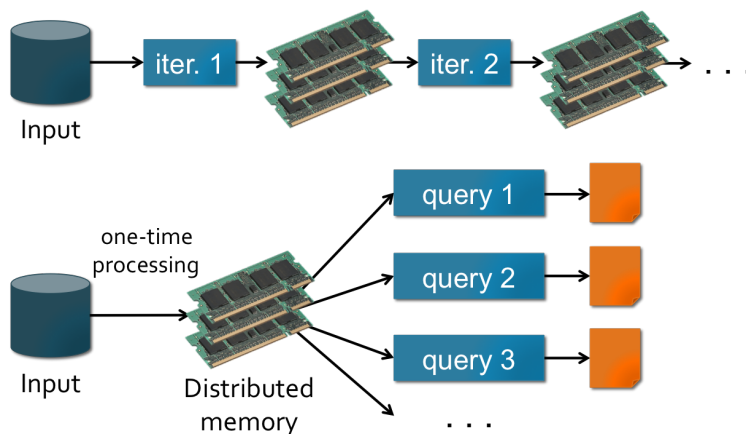
Question 2. What are the values v_j for the k^{th} iteration?

- A. The average PageRank value of the page j after the k^{th} step
- B. The probability that the surfer was at the node j at the $(k - 1)^{th}$ step
- C. The highest PageRank value of the page j after the k^{th} step
- D. The lowest PageRank value of the page j after the k^{th} step

3 Spark

MapReduce is great for batch processing, but users quickly needed to do more complex / multi-pass algorithms, and real-time stream processing. After MapReduce several systems were proposed for stream processing (including Apache Storm and Impala). However, the most used platform today is called Apache Spark (which is good for both iterative algorithms and on-line stream processing). Spark generalizes the MapReduce model to provide distributed processing over a shared-memory architecture. Unlike Hadoop MapReduce (which uses two-stage paradigm, which requires writing to disk after each Map and Reduce phase) Spark allows repeated access to the same data much faster thus offering several folds speed-up over Hadoop MapReduce. Note that Hadoop includes not just a storage component, known as the Hadoop Distributed File System, but also a processing component called MapReduce. Spark is equivalent to the Hadoop MapReduce but does not include its own distributed File System. Hence, it must integrate either with HDFS or another distributed file system such as Amazon S3.

Spark is generally a lot faster than MapReduce because of the way it processes data. While MapReduce operates in steps, Spark operates on the whole data set. The MapReduce workflow looks something like this: read data from the HDFS, perform an operation, write results to HDFS, read updated data from HDFS, perform next operation, write next results to HDFS, etc. Spark, on the other hand, completes the full data analytics operations in-memory and in near real-time: Read data from HDFS, perform all of the analytic operations, write results to HDFS. Hence, Spark can be as much as 10 times faster than MapReduce for batch processing and up to 100 times faster for in-memory analytics.



Question What is an RDD? Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset/record in an RDD is divided into logical partitions, which may be computed on different nodes of the cluster.

Question What operations can be applied to an RDD? Actions and Transformations. RDDs support two types of operations: transformations, which create a new dataset from an existing one, and actions, which return a value to the driver program after running a computation on the dataset. For example, map is a transformation that passes each dataset element through a function and returns a new RDD representing the results. On the other hand, reduce is an action that aggregates all the elements of the RDD using some function and returns the final result to the driver program (although there is also a parallel reduceByKey that returns a distributed dataset).

Question Are you able to write snippets of Spark (using python) to get the frequency of words in a set of documents?

4 Stream Processing

We covered two main algorithms used for stream processing, namely, BloomFilter and the Count-MinSketch. Be sure you understand these two algorithms.

5 Machine Learning

Big Data processing is composed of several phases: Data gathering, data cleaning and processing, data exploration, learning algorithm, and data summarization/visualization, and finally data storage.

Given the size and heterogeneity of the data, machine learning is applied to find hidden trends and rules in the data. We will discuss supervised and un-supervised learning algorithm:

- Supervised: All training samples are labeled.
- Unsupervised: Training is carried on data which is un-labeled. Must cluster the input data into classes on the basis of their statistical properties only.

Question We discussed K-Means, Classification, Regression (Linear Regression and Logistic Regression), and Naive Bayes algorithms in class. Are you able to briefly explain the various algorithms and identify whether they are a supervised or unsupervised learning task?

5.1 Regression

Lets start by talking about a few examples of supervised learning problems. Suppose we have a dataset giving the living areas and prices of 47 houses from Claremont, CA:

Living area	Price
2104	400
1600	350
2200	450
...	...

Given data like this, how can we learn to predict the prices of other houses in Claremont, as a function of the size of their living areas?

To describe the supervised learning problem slightly more formally, our goal is, given a training set, to learn a function $h : X \rightarrow Y$ so that $h(x)$ is a ‘good’ predictor for the corresponding value of y . For historical reasons, this function h is called a hypothesis. When the target variable that were trying to predict is **continuous**, such as in our housing example, we call the learning problem a **regression** problem. When y can take on only a small number of **discrete values**, we call it a **classification** problem.

Linear Regression is a regression problem, while Logistic regression is a classification problem.

Lets first discuss linear regression. Our goal is to find function $h(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$, where n is the number of features and θ are the parameters (also called weights) parameterizing the space of linear functions mapping from X to Y .

Now, given a training set, how do we pick, or learn, the parameters θ ? One reasonable method seems to be to make $h(x)$ close to y , at least for the training examples we have. To formalize this, we will define a function that measures, for each value of the θ s, how close the $h(x)$'s are to the corresponding y 's. We define the cost function: $J(\theta) = \frac{1}{2} \sum_{i=1}^m (h_{\theta}(x^{(i)}) - y^{(i)})^2$

The basic idea is that we want to choose θ so as to minimize $J(\theta)$. To do so, let's use a search algorithm that starts with some 'initial guess' for θ , and that repeatedly changes θ to make $J(\theta)$ smaller, until hopefully we converge to a value of θ that minimizes $J(\theta)$. This is done using the gradient descent algorithm. Look at the slides for more details on gradient descent.

Logistic regression on the otherhand is a classification task. This is just like the regression problem, except that the values y we now want to predict take on only a small number of discrete values. For now, we will focus on the binary classification problem in which y can take on only two values, 0 and 1.

Since it doesn't make sense for our hypotheses function for this task to take values larger than 1 or values smaller than 0, we should change the form of the hypotheses $h_{\theta}(x)$ to $h_{\theta}(x) = g(\theta^T x) = \frac{1}{1 + \exp^{-\theta^T x}}$, where $g(z) = \frac{1}{1 + \exp^{-z}}$ is the logistic function or sigmoid function. The sigmoid function only gives values between 0 and 1 which is appropriate for our problem. We can solve this problem using gradient descent with our new hypothesis function. See slides for details.

5.2 Naive Bayes

Naive Bayes is a supervised classification problem that is generally known for being a 'extremely fast' classification algorithm. The classification technique is based on Bayes' Theorem with an assumption of independence among the predictors.

Bayes theorem provides a way of calculating posterior probability $P(C|x)$ from $P(C)$, $P(x)$ and $P(x|c)$.

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

The diagram shows the formula with arrows pointing from labels to terms: 'Likelihood' points to $P(x|c)$, 'Class Prior Probability' points to $P(c)$, 'Posterior Probability' points to $P(c|x)$, and 'Predictor Prior Probability' points to $P(x)$.

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \dots \times P(x_n|c) \times P(c)$$

Where,

- $P(C|x)$ is the posterior probability of class (c , target) given predictor (x , attributes).
- $P(C)$ is the prior probability of class.
- $P(x|C)$ is the likelihood which is the probability of predictor given class.
- $P(x)$ is the prior probability of predictor.

So, for a new datapoint x , we simply compute the various $P(C_i|x)$ for each class i , and choose the class with highest probability. But .. but.. how do we compute $P(C)$, $P(x|C)$, and $P(x)$? Well, the answer is simple, we extract this information from the data.

Weather	Play
Sunny	No
Overcast	Yes
Rainy	Yes
Sunny	Yes
Sunny	Yes
Overcast	Yes
Rainy	No
Rainy	No
Sunny	Yes
Rainy	Yes
Sunny	No
Overcast	Yes
Overcast	Yes
Rainy	No

So, next we need to compute the likelihood table.

Probability of the class:

$$P(\text{No}) = 5/14 = 0.36$$

$$P(\text{Yes}) = 9/14 = 0.64$$

Probability of the features:

$$P(\text{Overcast}) = 4/14 = 0.29$$

$$P(\text{Sunny}) = 5/14 = 0.36$$

$$P(\text{Rainy}) = 5/14 = 0.36$$

Problem: Will the Dodgers play the Astros if the weather is sunny?

So, we would need to compute the probability of $P(\text{YES}|\text{Sunny})$ and $P(\text{NO}|\text{Sunny})$.

$$P(\text{Yes}|\text{Sunny}) = \frac{P(\text{Sunny}|\text{Yes}) * P(\text{YES})}{P(\text{SUNNY})}$$

$$P(\text{No}|\text{Sunny}) = \frac{P(\text{Sunny}|\text{No}) * P(\text{No})}{P(\text{SUNNY})}$$

We need to compute the $P(\text{Sunny}|\text{Yes})$ which is $3/9=0.33$, and $P(\text{Sunny}|\text{No})$ which is $2/5=.4$

$$P(\text{Yes}|\text{Sunny}) = \frac{P(\text{Sunny}|\text{Yes}) * P(\text{Yes})}{P(\text{Sunny})} = \frac{0.33 * 0.64}{0.36} = 0.60$$

$$P(\text{No}|\text{Sunny}) = \frac{P(\text{Sunny}|\text{No}) * P(\text{No})}{P(\text{Sunny})} = \frac{0.40 * 0.36}{0.36} = 0.40$$

So, yes, they should play given the weather is sunny. The approach is the same if we have more than one feature.

5.3 Clustering

Clustering is the grouping of a particular set of objects based on their characteristics, aggregating them according to their similarities. Clustering is an unsupervised learning task and groups points such that points in the same cluster are very similar to each other, and points in different clusters are dissimilar to each other.

There is generally two approaches to clustering: hierarchical and partitional (kmeans is one partitional approach).

Question Usually we must represent the cluster center. How do we do this in euclidean space? How do we do this in non-euclidean space?

Well, we can either represent clusters using a centroid or a clustroid. Centroid is the average of all (data) points in the cluster. This means centroid is an ‘artificial’ point. Clustroid is an existing (data) point that is ‘closest’ to all other points in the cluster.

QUESTION Can you describe the kMeans algorithm.

Given k , the k-means algorithm works as follows:

1. Choose k (random) data points (seeds) to be the initial centroids, cluster centers
2. Assign each data point to the closest centroid
3. Re-compute the centroids using the current cluster memberships
4. If a convergence criterion is not met, repeat steps 2 and 3

Question Are you able to describe the heuristics we can apply to deal with (1) outliers, (2) ‘bad’ initial cluster centers, (3) choosing a good k ?

5.4 Validation

A common problem with machine learning is ‘over-fitting’. Over-fitting refers to learning from the ‘data’ rather than learning the underlying function that correctly predicts the data model.

Over-fitting can be identified when the training model performs well on the training data, but performs poorly with new datasets (unseen data). Hence, we need to make sure we have a sufficiently large dataset to train the algorithms. Moreover, the training dataset is usually divided into 3 sets (as described below) to verify that overfitting is avoided and the model is tuned effectively.

- Training set: a set of examples used for learning (either supervised or unsupervised learning).
- Validation set: a set of examples used to tune the architecture of a classifier and estimate the error.
- Test set: used only to assess the performances of a classifier. It is never used during the training process so that the error on the test set provides an unbiased estimate of the generalization error

The above method is generally referred to as the holdout method. Another validation method is called K-fold cross validation. K-fold cross validation is one way to improve over the holdout method. The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other $k-1$ subsets are put together to form a training set. Then the average error across all k trials is computed. The advantage of this method is that it matters less how the data gets divided. Every data point gets to be in a test set exactly once, and gets to be in a training set $k-1$ times. The variance of the resulting estimate is reduced as k is increased. The disadvantage of this method is that the training algorithm has to be rerun from scratch k times, which means it takes k times as much computation to make an evaluation. A variant of this method is to randomly divide the data into a test and training set k different times. The advantage of doing this is that you can independently choose how large each test set is and how many trials you average over.

5.5 Dimensionality Reduction

The curse of dimensionality refers to various phenomena that arise when analyzing and visualizing data in high-dimensional spaces. Dimensionality Reduction :

- reduces the time and storage space required for our Machine Learning algorithms.
- Removal of multi-collinear features improves the performance of the machine learning model.

- It becomes easier to visualize the data when reduced to very low dimensions such as 2D or 3D.

The main idea behind dimensionality reduction is to reduce the space of the data. The main idea is to project the d -dimensional data points to a k -dimensional space, where k is much smaller than d and distances in the low-dimensional embedding are still preserved as well as possible. The problem is then re-examined in the lower dimensions allowing for complex machine learning algorithms to be applied.

QUESTION What is the name of the algorithm/approach we discussed for dimensionality reduction, and what is the basic idea behind this approach ?

5.6 Recommender Systems

We covered two types of recommender systems:

- Content-based systems : examine the properties of the items liked by the user and recommend similar items. Similarity of items is determined by measuring the similarity in their properties.
- Collaborative filtering systems : recommend items based on similarity between users and/or items.

The advantages/disadvantages of a collaborative-based system includes:

Pros	Cons
No need for data on other users	Finding features is hard
No cold-start or sparsity problems	Unable to recommend to new users
Able to recommend to users with unique tastes	Overspecialization
Able to recommend new & unpopular items	
Able to provide explanations	

The advantages/disadvantages of a content-based system includes:

Pros	Cons
Feature extraction is not needed	Cold Start: Need enough users in the system to find a match
	Sparsity
	Popularity Bias

QUESTION We covered basically two approaches for applying recommender systems, KNN and ALS. Do you understand the basics behind these two methods?

5.7 Anomaly Detection

Anomaly detection (also outlier detection) is the identification of items, events or observations which do not conform to an expected pattern or other items in a dataset.

Anomaly detection differs from supervised learning because usually when applying anomaly detection the dataset has very few examples of anomalies. If the dataset is quite large and has an equal number of positive and negative examples, we can simply apply supervised learning methods. However, if we find that the number of positive examples (anomalies) is small, then we must apply an anomaly detection approach like 'Multivariate Gaussian Distribution' to detect anomalies. In such an approach, we try to learn the model $p(x)$ of the negative examples. When given a new example y , we compare $p(y)$ to a pre-determined threshold to determine whether y is an anomaly or follows the normal distribution. Look at the slides for more details.

6 NoSQL Databases





The Hadoop Distributed File System (HDFS) was not sufficient for the many transaction-based application that increase had to process, store, query, and manage Big Data. Moreover, commercial databases were not sufficient for these applications, because: (1) most commercial databases cannot scale to the amount of data being processed/queried, (2) cost of specialized databases is very high (many small companies were dealing with Big Data), and (3) much of this data was semi-structured or raw logs, and did not translate well into relational data. Since, traditional databases were not a viable option, the 'NoSQL' movement started.

What is NoSQL? It stands for 'Not Only SQL' and such database solutions usually deal with :

- Large amounts of data (peta bytes) that require scalability of a cluster
- Data in various formats (raw logs, semi-structured, graph, etc.
- Sparse dataset with no-schema or ever-changing schema

6.1 Data Model

Several NoSQL databases were proposed, and they can be classified based on their underlying data model as follows:

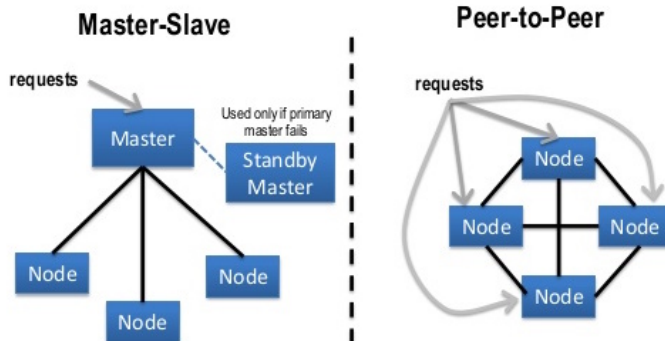
<i>Types of NoSQL DBs</i>	
Key-Value Databases	
Column-Family Databases	
Document Databases	
Graph Databases	

- Key-Value : Basically a large hash-map. The key-value model is the simplest and easiest to implement, but it is inefficient when you are only interested in querying or updating part of a value.
- Column-Family : A more complex data model compared to the Key-Value data model. A key is mapped to multiple columns, arranged by a column-family.
- Document-based : Stores entire documents by key, which can contain nested documents. Allows use of indexes to speed-up lookups.
- Graph-based : A large graph that represents the nodes and their inter-connection... used for social network modeling.

6.2 Data Distribution

Beyond the data model, the NoSQL databases must distribute data across the cluster in-order to scale. There are two styles of distributing data. In both policies, replication of the data accross multiple servers is important is important to ensure fault tolerance.

Master-slave replication makes one node the authoritative copy that handles writes while slaves synchronize with the master and may handle reads. **Peer-to-peer** replication allows writes to any node; the nodes coordinate to synchronize their copies of the data. Master-slave replication reduces the chance of update conflicts but peer-to-peer replication avoids loading all writes onto a single server creating a single point of failure.



6.3 CAP theorem

In a distributed system, managing consistency(C), availability(A) and partition toleration(P) is important, Eric Brewer put forth the CAP theorem which states that in any distributed system we can choose only two of consistency, availability or partition tolerance. Many NoSQL databases try to provide options where the developer has choices where they can tune the database as per their needs. For example if you consider Cassandra a distributed column-oriented database. There are essentially three variables r , w , n where

1. r =number of nodes that should respond to a read request before its considered successful.
2. w =number of nodes that should respond to a write request before its considered successful.
3. n =number of nodes where the data is replicated aka replication factor.

