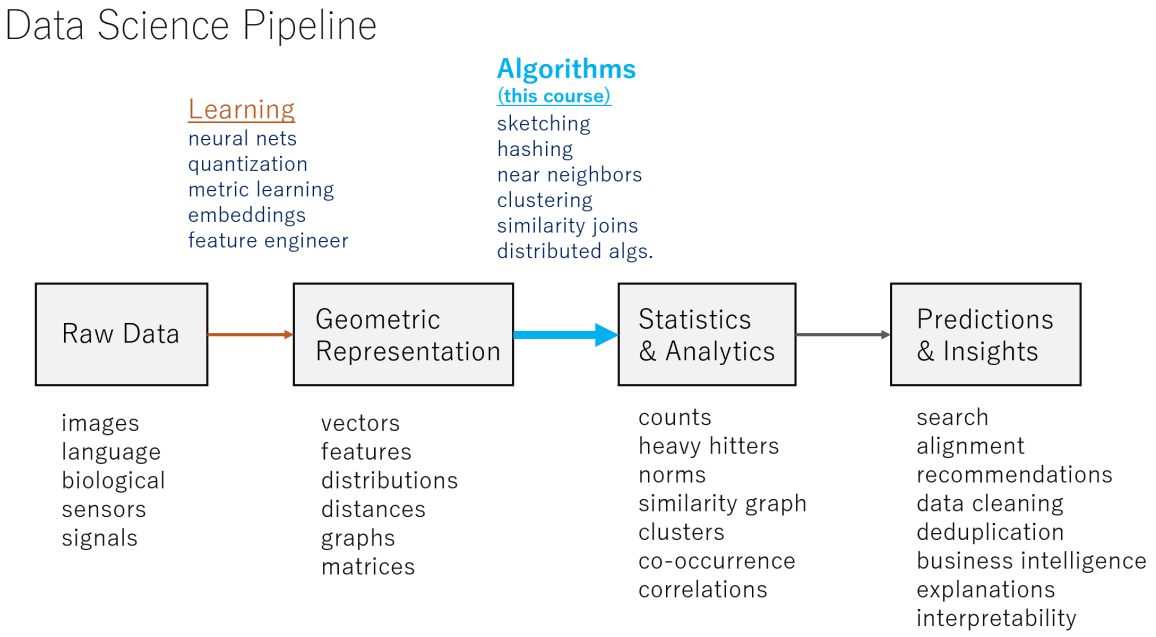
**Algorithms for Big Data** Fall 2020

# Lecture 01 — October 2, 2020

*Prof. Cyrus Rashtchian* *Topics: Overview*




## 1 Topics Covered in the Course

The above figure motivates this course by placing the topics in a typical ML or data analytics pipeline. The point is that geometric data arises everywhere, and the many advances in (deep) learning means that the best data representations tend to be geometric (e.g., vectors and distances). After this layer of learning, the task is to quickly process/analyze the resulting data. For huge datasets, this motivates the need for efficient algorithms. We will focus on popular techniques that come with *provable guarantees*. For example, if the pipeline has many steps, we will want some certainty that the intermediate steps will always work very well! The unifying theme is that all of the course topics deal with algorithms for data with geometric structure. The focus is on approximation algorithms that scale to handle large datasets in modern systems (a.k.a., *big data*). In contrast to standard algorithms courses, we aim to to handle situations where a data set is so large or a computation is so complex that a naive approach would require an impractical amount of memory or time. We will often exploit geometric structure and/or use randomized algorithms to achieve the improved bounds. We will cover the following core topics:

1. sketching / streaming
2. dimensionality reduction
3. nearest neighbor search
4. clustering
5. special topics: distributed algorithms, DNA storage, adversarial robustness

## Topic 1: Sketching and Streaming

Computing devices have become smaller (e.g., smart phones, smart watches) and more common in real-world environments (e.g., adaptive thermostats, digital voice assistants). The mobility of these devices enables new and exciting experiences, especially because of machine learning. Unfortunately, these devices have stringent resource constraints, including reduced processing capabilities, decreased power supplies, and bounded internet connections. Hence, it is challenging to create programs that run quickly and reliably on these devices. The common methods for machine learning and algorithms that run on a laptop would consume a prohibitive amount of resources.

How do we design new algorithms that adhere to resource constraints without compromising their functionality? Theoretical researchers have developed a mathematical abstraction of efficient programs that unifies the way we analyze various constraints. This allows them to better understand the fundamental tradeoffs among these real-world resources. It also facilitates the development of machine learning methods that are aware of, and optimally adapt to, the resources available. Finally, these advances have helped reduce the energy consumption of standard computers by using a similar tradeoff analysis.

A central theme will be trading-off accuracy for a reduction in space. This can be thought of as an algorithmic analogue of lossy compression. We study *sketching*, which corresponds to a compression function $C$. This compresses a data set $x$ into a different representation $C(x)$. The aim is to make $C(x)$ as small as possible, while still enabling us to query $f(x)$ when given access to only $C(x)$, as opposed to all of $x$ (e.g., when $x$ is too large to store/process). We will see many examples: approximate counting/histograms, estimating the number of distinct elements, and norm estimation.

There are some things we might want when we are designing such a sketch/compression $C(x)$. Perhaps, we want $f$ to take on two arguments instead of one, as is the case when we want to compute $f(x, y)$ from $C(x)$ and $C(y)$. Often, we want $C(x)$ to be composable. In other words, if $x = x_1x_2x_3 \dots x_n$, we want to be able to compute $C(xx_{n+1})$ using just $C(x)$ and $x_{n+1}$.

A main use of sketching techniques is for *streaming* algorithms (think physical sensors or ever-growing social media data like tweets/posts). A stream is a sequence of data elements that comes in bit by bit (or number by number, etc), like items on a conveyor belt. Streaming is the act of processing these data elements *on the fly* as they arrive (as opposed to computing something in batch after loading the whole dataset). The goal of streaming is to answers queries within the constraints of sublinear memory. We can (and will) use sketches to design streaming algorithms.

## Topic 2: Dimensionality Reduction

There are many instances where data sets have high dimensionality. For example, consider spam filtering. A simple approach to spam detection uses a bag-of-words model, where each email is represented as a high-dimensional vector. The indices come from a dictionary of words, and the value at each index is the number of occurrences of that word. In situations like these where we have a high-dimensional computational geometry problem, we may want to reduce the number of dimensions in pre-processing while preserving the approximate geometric structure. This speeds up the eventual algorithm because we can process data using the lower dimensional data.

## Topic 3: Nearest Neighbor Search and Locality Sensitive Hashing

We consider two algorithmic problems for geometric data: nearest neighbor search and clustering. For the first problem, given a query, the goal is to find the nearest point(s) in a dataset to the query.

A simple way to do this is a linear scan. If there are $n$ elements in $d$-dimensional space, we can store all of them using $O(nd)$ space. Then, on a query $q$, we can compare $q$ to all $n$ vectors in the dataset. This takes time $O(nd)$. Can we do better?

Yes, if we are okay allowing an approximation. Then, we can improve this to sublinear time by increasing the space a little bit. The main technique will be locality sensitive hashing (LSH). The idea to develop noisy hash functions where the query will be more likely to hash to the same buckets as similar input vectors (and less likely for far/dissimilar inputs). Beyond nearest neighbor search, LSH-related ideas are prevalent and hugely important in many many practical applications (image search, AR/VR, NLP, bioinformatics, etc).

## Topic 4: Clustering

Clustering is another fundamental geometric problem. The goal is to find a small number of clusters that summarize or group the data into meaningful subsets. This is often hard to define precisely, and a good clustering will often depend on the data or the domain. We will consider multiple ways of doing this, with different objectives and guarantees. For example $k$-means, $k$-median, and $k$-center all will find $k$ center points to minimize some cost function. This cost will change, depending on how we measure the distance.

The bad part about clustering is that most objectives are NP-Hard. The good (and mysterious) part is that people use heuristics all the time (in all areas of science) and things mostly work out.

## Special Topics

**Explainable ML.** Recently, there has been a lot of interest in better understanding exactly how machine learning algorithms work. For example, neural networks are very good at classification, but sometimes it can be very mysterious why they make the predictions that they do. This course covers clustering already, so we will look at explainable ML in the context of unsupervised learning. Many clustering algorithms are very global. They iteratively change the cluster assignments based on the current cost or other objectives, and they converge over time to a solution. The issue is that in the end, it's not exactly clear why certain points have been assigned to their clusters. Is it because of specific features or is it just because somehow the current clusters are better than other ones? We can formally study this question by asking how well we can cluster with a decision tree. Each node has a feature-threshold pair, and so the tree gradually partitions the dataset one feature at a time. Then, the leaves are labeled with cluster IDs. This way we can characterize the cluster assignments by looking at the paths in the tree to the leaves (for each data point, there will be some path to a leaf, which explains why the point is in the cluster). Surprisingly, we can also get a pretty good bound on the $k$-means and $k$-medians cost when we cluster in this way.

**Distributed Algorithms.** If a data set is really large, it may not be possible to store or process all the data at the same time using only one processor. One reason is that algorithms are very slow if the data is not in main memory (RAM/Cache). So it is often better to use a large number of processors/cores, either with a cluster of CPUs or GPUs. But this makes the algorithm design process more complicated — and more interesting! We consider distributed algorithms for geometric problems, such as similarity search and clustering. A nice aspect of algorithms in this area is that there is a common framework to design/analyze algorithms. It is based on MapReduce. Even though the specifics of systems matter, there are also basic metrics that determine the quality/speed of an algorithm. For example, we will consider the total communication among processors, and we will trade this off with the amount of work that each processor must do.

**Robustness.** Classifiers are known to be vulnerable to adversarial examples. Such examples are just a fancy name for imperceptible modifications of true inputs that lead to misclassification. This raises many concerns, and recent research aims to better understand this phenomenon. We will briefly survey some geometric ideas in this area and how they relate to some of the other topics we have talked about. This area is fairly new, and we need better methods and better theory around this topic. The cool thing is that a lot of the current methods are very geometric, and hence, they fit well within the theme of the course. There has been work on making the $k$-NN classifier more robust as well, which is related to some of the topics we will cover regarding approximate nearest neighbor search. Finally, for neural networks, we will also see that training the network to be more robust will be essentially equivalent to finding geometric constraints on the decision regions of the neural network. We want the predictions to remain constant even with small perturbations, so we want to modify the loss function or regularization to encourage this. Unfortunately, it's still a very big open question how to ensure both robustness and high test accuracy on normal inputs, which is what we really want from a robust classifier.