Data Science Through a Geometric Lens

UCSD CSE 291 (F00) Fall 2019

- **Instructor:** Cyrus Rashtchian
- **Lectures:** Monday + Wednesday 10:30a – 11:50a
- **Location:** CSE Building, Room 4140 (4th floor, end of the hallway, on the left)
- **Course Website:** http://madscience.ucsd.edu/cse291.html

Description: This is an advanced algorithms course, with a focus on geometric algorithms motivated by data science applications. Many applications (computer vision, AR/VR, recommender systems, computational biology) rely on geometric ideas and data representations. In this context, geometry refers to distances between high-dimensional vectors (e.g., Lp distances) or strings (e.g., edit distance). This course will provide the foundations for designing and analyzing algorithms operating on data with geometric structure. We will begin with sketching/streaming algorithms and dimensionality reduction. Then, we will explore many algorithmic problems such as clustering and approximate nearest neighbor search via locality sensitive hashing. Finally, we will address massive data sets by adapting algorithms to distributed models. Students will be exposed to many open research problems.

Topics: We will cover five main vignettes (each 3-4 classes):

1. Sketching / Streaming for Big Data
2. Dimensionality Reduction and Metric Embeddings
3. Approximate Nearest Neighbor Search and Locality Sensitive Hashing
4. Clustering
5. Distributed Algorithms in a MapReduce Model

Required Knowledge: Algorithms, linear algebra, and probability theory. Mathematical maturity is a must: the class is based on theoretical ideas and is proof-heavy. You are expected to be able to read and write formal mathematical proofs. Some familiarity with algorithms and randomness will be assumed (exposure to graduate-level algorithms is a plus).

Recommended Preparation: Review basic linear algebra (vector norms, inner products, orthogonality) and probability theory (vectors of random variables, pairwise independence, Chernoff bounds).

Supplementary Reading: There is no official textbook for the course. The following references contain parts of the material that will be covered in class.

- *Mining of Massive Datasets* by Jure Leskovec, Anand Rajaraman, and Jeff Ullman.
  Online copy available at http://i.stanford.edu/~ullman/mmdsn.html
- *Foundations of Data Science* by Avrim Blum, John Hopcroft, and Ravindran Kannan.
  Online copy available at https://www.cs.cornell.edu/jeh/book.pdf

Course Expectations

Course Scope: The course will be mostly theoretically, but students should not shy away from working on empirical aspects of the material. For example, the final project may be an entirely practical/empirical undertaking. Algorithms can be developed that exploit everything from software libraries to specific architectures to dataset properties. All of this is important and relevant for modern data science. The problem sets will have a mixture of easier and harder theoretical questions, and at least one implementation question.

Evaluation: There will be 3 problem sets and a final project (no exams). The problems sets will have a mixture of theory and implementation questions. The project will be completed in groups of 1-3, and the goal is to take on a research-level investigation (theoretical or empirical or a mixture of both). The grades will be **based 50% on the homework and 50% on the final project, where the final project itself will have 5% of the final grade based on project proposal (due 11/6), and 10% based on the presentation (on 12/2 or 12/4), and 35% based on the project report (due 12/5).**

Homework Grading: Each problem set will consist of many questions, where the students will be asked to solve a subset of questions (for example, complete 3 of 5 homework problems). The intention is that the students will choose problems to their liking, that fit their background. For theory questions, the requirement is always to provide a formal argument and proof (see below for writing up). For implementation questions, the requirement is to implement the indicated algorithm, and demonstrate certain properties of the algorithm. The results should be given both as pseudo-code and with graphs/plots to demonstrate different trade-offs (for example, time/memory versus accuracy). You must clearly identify yourselves at the top of each homework, and you must make it clear which questions you are answering. It is ideal and preferred if you type up the solutions in Latex (see below for details).

Writing up solutions: You must produce precise and formal proofs. The goal of the class, in part, is for you to learn to reason about algorithms, precisely describe them, and formally prove claims about their correctness and performance. Hence, it is important that you write up your assignments clearly, precisely, and concisely. Legibility of your write-up will be an important factor in its grading. When writing up (algorithmic) solutions, keep in mind the following:

- The best way for you to convey an algorithm is by using plain English description. A worked example can also help; but revert to pseudocode only if necessary. Generally, give enough details to clearly present your solution, but not so many that the main ideas are obscured.
- The analysis of the algorithm must include both 1) proof of correctness, and 2) formal upper bound on performance (usually runtime, but sometimes space as well). This is true for theoretical and implementation questions.
- You are encouraged (but not required) to type up your solutions using LaTeX. Latex is the standard package for typesetting and formatting mathematically-rich content. Since LaTeX knowledge is a good life skill, now may be a good chance to learn it. A short mini-course on LaTeX is available: http://www.ctan.org/tex-archive/info/lshort/english/lshort.pdf. Macros to format pseudocode are available at http://www.cs.dartmouth.edu/~thc/clrscode/
- Lectures will generally be at a slightly lower level of formalism, in the interest of time.

Project Grading: The project can either be an in-depth exploration of a paper or a new research-level investigation. Overall, the final project does not need to be original to the level of a research publication, but the effort of the students should be clear. On the other hand, students are encouraged to work on projects that may turn into future publications at research venues. For theory projects, the goal is to explore the possibilities and limitations of known techniques (ideally improving known results and/or expanding knowledge to new regimes). For empirical projects, the goal is to study the efficacy and trade-offs of algorithms in specific settings (either on standard datasets or on compelling synthetic data distributions). The ideal outcome of a project consists of new insights and/or examples that shed light on previously studied problems. The following types of projects are encouraged:

- Reading-based: read a few recent research papers on a concrete topic and summarize them.
- Implementation-based: implement some of the algorithms from the class (or from other theoretical literature), and perhaps apply to your area of interest/expertise, using real-world datasets. One aspect of such projects must be comparison among a few algorithms.
- Research-based: investigate a research topic on your own (e.g., develop an algorithm, and prove its properties; or prove an impossibility result). It may be more applied: e.g., perhaps in your area, certain theoretical algorithms can be modified to have even better performance, due to special properties of the datasets, etc.

The topic of your project must be within the scope of Theoretical Computer Science, Data Science, or Machine Learning (please talk to me otherwise). It is expected that there will be an algorithmic component. In particular, the focus is on algorithms with provable guarantees (for the implementation type, you may compare such theoretical guarantees with heuristics though).

Late Policy: Everyone has a default 3 days of extension (fractions of a day are rounded up), over all the homeworks. Once you've used up the 3 days, late homeworks will be penalized at the rate of 10%, additively, per late day or part thereof (i.e. fractions of a day are rounded up), for up to 7 days. To allow us to distribute the solutions in a timely fashion, homeworks submitted more than 7 days after the deadline will not be accepted. Exceptions will be made only for exceptional unforeseen circumstances (e.g., serious illness), in which case you will need to provide some additional documentation (e.g., doctor's note). You are strongly encouraged to start working on the homeworks early: some problems may require you to sit on the problem for a while before you get your "aha" moment. Starting early also gives you time to ask questions and make effective use of the office hours of the teaching staff.

External Resources: Aim to solve the problems on your own, using simply the course lecture notes and standard resources (e.g., books on algorithms, probability theory, or linear algebra). In many cases, it will be possible to find solutions in papers or in other courses. If you use external resources to inform your solution, you must cite the relevant source. Moreover, you must provide a self-contained and complete solution to the homework problem. In other words, it is okay to look at other resources for inspiration, but all the writing must be in your own words. You must feel confident that what you hand in represents your current knowledge. Direct copying is not allowed. For the final project, a significant portion of the work (that is, the majority) should be based on original findings or original discussion.

Group work: For homework, students may work in groups of size 1-3. You should hand in one assignment for each group. You must clearly list all students consulted during the completion of the assignment. For the project, you may work in groups of size 1-3. It is ideal to work in groups of size 2-3. The goal of the homework and projects is to collaborate and to learn how to excel at group work.