Data contains value and knowledge
Data Mining & Machine Learning

- But to extract the knowledge data needs to be
  - Stored (systems)
  - Managed (databases)
  - And ANALYZED ← this class

Data Mining ≈ Big Data ≈ Predictive Analytics ≈ Data Science ≈ Machine Learning
What This Course Is About

- **Data mining** = extraction of actionable information from (usually) very large datasets, is the subject of extreme hype, fear, and interest

- It’s not all about machine learning
- But some of it is

- Emphasis in CS547 on algorithms that **scale**
  - Parallelization often essential
Data Mining Methods

- **Descriptive methods**
  - Find human-interpretable patterns that describe the data
  - **Example:** Clustering

- **Predictive methods**
  - Use some variables to predict unknown or future values of other variables
  - **Example:** Recommender systems
This Class: CS547

- This combines best of machine learning, statistics, artificial intelligence, databases but emphasis on
  - **Scalability** (big data)
  - **Algorithms**
  - **Computing architectures**
  - Automation for handling large data
What will we learn?

- We will learn to **mine different types of data:**
  - Data is high dimensional
  - Data is a graph
  - Data is infinite/never-ending
  - Data is labeled

- We will learn to **use different models of computation:**
  - MapReduce
  - Streams and online algorithms
  - Single machine in-memory
What will we learn?

- **We will learn to solve real-world problems:**
  - Recommender systems
  - Market Basket Analysis
  - Spam detection
  - Duplicate document detection

- **We will learn various “tools”:**
  - Linear algebra (SVD, Rec. Sys., Communities)
  - Optimization (stochastic gradient descent)
  - Dynamic programming (frequent itemsets)
  - Hashing (LSH, Bloom filters)
How the Class Fits Together

<table>
<thead>
<tr>
<th>High dim. data</th>
<th>Graph data</th>
<th>Infinite data</th>
<th>Machine learning</th>
<th>Apps</th>
</tr>
</thead>
<tbody>
<tr>
<td>Locality sensitive hashing</td>
<td>PageRank, SimRank</td>
<td>Sampling data streams</td>
<td>SVM</td>
<td>Recommender systems</td>
</tr>
<tr>
<td>Clustering</td>
<td>Network Analysis</td>
<td>Filtering data streams</td>
<td>Decision Trees</td>
<td>Association Rules</td>
</tr>
<tr>
<td>Dimensionality reduction</td>
<td>Spam Detection</td>
<td>Queries on streams</td>
<td>Perceptron, kNN</td>
<td>Duplicate document detection</td>
</tr>
</tbody>
</table>
Course Logistics
Course Staff

Ken Gu  
(Head TA)

Yikun Zhang

Esteban Safranchik

CS547 Course Staff

- **Office hours:**
  - See course website [www.cs.washington.edu/cse547](http://www.cs.washington.edu/cse547) for TA office hours
    - *We start Office Hours today*
  - **Tim:** Tuesdays 11:20-12:00pm (right after lecture)
  - **TA office hours:** see website and calendar
Resources

- **Course website:** [www.cs.washington.edu/cse547](http://www.cs.washington.edu/cse547)
  - Lecture slides (at least 30min before the lecture)
  - Homeworks, readings

- **Need for accessible course materials?** Let us know!

- **Class textbook:** *Mining of Massive Datasets* by A. Rajaraman, J. Ullman, and J. Leskovec
  - Sold by Cambridge Uni. Press but available for free at [http://mmds.org](http://mmds.org)
  - Course based on textbook and Stanford CS246 course by Leskovec and others
Logistics: Communication

- **Ed Q&A website:**
  - https://edstem.org/us/courses/32320/discussion/
  - Use Ed for **all questions** and public communication & announcements
    - Search the forum before asking a question
    - Please tag your posts and please no one-liners

- (Only) for personal matters, email course staff at:
  - cse547-instructors@cs.washington.edu

- We will post course announcements to Ed (make sure you check it regularly)
Special Tutorials

- **Spark tutorial and help session:**
  - Thursday, Jan 5, 3:30-5pm CSE2 371

- **Review of basic probability and proof techniques**
  - Tuesday, Jan 10, 3:30-5pm CSE2 371

- **Review of linear algebra:**
  - Thursday, Jan 12, 3:30-5pm CSE2 371
Work for the Course: Homworks

- **4 longer homeworks:** 40%
  - Four major assignments, involving programming, proofs, algorithm development.
  - We improve homeworks every year and strive to give you well-defined problems that maximize your learning and minimize your time spent. Sometimes this means lots of instructions. Don’t worry – this is there to help you.
  - Assignments take lots of time (+20h). **Start early!!**

- **How to submit?**
  - **Homework write-up:**
    - Submit via Gradescope
    - Course code: see website
  - **Everyone uploads code:**
    - Put all the code for 1 question into 1 file and submit via Gradescope
Work for the Course: Colabs

- **Short weekly Colab notebooks: 20%**
  - “Colab” is a **free cloud service from Google**, hosting Jupyter notebooks with free access to GPU and TPU
  - Colab notebooks are posted every **Thursday**
    - 10 in total, from 0 to 9, each worth 2%
  - Due one week later on **Thursday 23:59 PST. No late days!**
    - First 2 Colabs will be posted on Thu, including detailed submission instructions to Gradescope (unlimited attempts)
    - Colab 0 (Spark Tutorial) will be **solved in real-time during Spark recitation session!**
  - Colabs require at most **1hr of work**
    - **few lines of code!**
# Homework Calendar

- **Homework schedule (without weekly Colabs)**

<table>
<thead>
<tr>
<th>Date (23:59 PT)</th>
<th>Released</th>
<th>Due</th>
</tr>
</thead>
<tbody>
<tr>
<td>01/02, Today</td>
<td></td>
<td></td>
</tr>
<tr>
<td>01/05, Thu</td>
<td>HW1 (and Colab 0/1)</td>
<td>HW1</td>
</tr>
<tr>
<td>01/19, Thu</td>
<td>HW2</td>
<td></td>
</tr>
<tr>
<td>01/26, Thu</td>
<td><em>Project Proposal</em></td>
<td></td>
</tr>
<tr>
<td>02/02, Thu</td>
<td>HW3</td>
<td>HW2</td>
</tr>
<tr>
<td>02/09, Thu</td>
<td><em>Project Milestone</em></td>
<td></td>
</tr>
<tr>
<td>02/16, Thu</td>
<td>HW4</td>
<td>HW3</td>
</tr>
<tr>
<td>03/02, Thu</td>
<td>HW4</td>
<td></td>
</tr>
<tr>
<td>03/12, Sun</td>
<td><em>Project Report</em></td>
<td></td>
</tr>
<tr>
<td>03/13, Mon</td>
<td><em>Project Presentation</em></td>
<td></td>
</tr>
</tbody>
</table>

- **Two late periods for HWs for the quarter:**
  - Late period expires 48 hours after the original deadline
  - Can use max 1 late period per HW (not for *Project / Colabs*)
Work for the Course: Course Project

- **Course Project:** 40%
  - Project proposal (20%)
  - Project milestone report (20%)
    - Why extra milestone? We added this so that we can give you meaningful feedback on your projects and help you learn.
  - Final project report (50%)
  - Project Presentation (10%)
  - *More details on course website*

- Teams of (up to) **three students each**
  - Start planning now
  - Find students in class, office hours, or through Ed
  - Find dataset to work on – also see course website
Work for the Course: Course Project

- **Project Presentation**
  - **Monday, March 13, 10:30am-12:20pm**
  - **You have to be present!**
  - **Location: CSE2 G10**
  - **Exact format will be announced on website**

- **Extra credit:** Up to 2% of your grade
  - For participating in Ed discussions
    - Especially valuable are answers to questions posed by other students on Ed, office hours or throughout lectures
  - Reporting bugs in course materials
  - See course website for details
Prerequisites

- **Programming**: Python
- **Basic Algorithms**: e.g., CS332/CS373 or CS417/CS421
- **Probability**: any introductory course
  - There will be a review session and a review doc is linked from the class home page
- **Linear algebra**: (e.g., Math 308 or equivalent)
  - Another review doc + review session is available
- **Rigorous proofs & Multivariable calculus**: (e.g., CS311 or equivalent)
- **Database systems** (SQL, relational algebra)
What If I Don’t Know All This Stuff?

- Each of the topics listed is important for a small part of the course:
  - If you are missing an item of background, you could consider just-in-time learning of the needed material. This will typically add to your workload!

- The exception is programming:
  - To do well in this course, you really need to be comfortable with writing code in Python
Collaboration Policy & Academic Integrity

- We’ll follow the standard CS Dept. approach: You can get help, but you **MUST** acknowledge the help on the work you hand in

- Failure to acknowledge your sources is a violation of academic integrity

- We use plagiarism tools to check the originality of your code
Collaboration Policy & Academic Integrity

- You can talk to others about the algorithm(s) to be used to solve a homework problem;
  - As long as you then mention their name(s) on the work you submit. You still need to come up with your own write-up. Don’t just copy it!
- You should not use code of others or be looking at code of others when you write your own:
  - You can talk to people but have to write your own solution/code
  - If you fail to mention your sources, plagiarism tools or TAs will catch you, and you will be charged with an academic integrity violation.
Final Thoughts

- **CS547 is fast paced!**
  - Requires programming maturity
  - Strong math skills
    - Some students tend to be rusty on math/theory
- **Course time commitment:**
  - Homeworks every two weeks that take +20h
  - Significant course project
- Form study groups
- Form project groups

- It’s going to be **fun** and **hard** work. 😊
5 To-do items

- **5 to-do items for you:**
  - Make sure you can access Canvas & Ed
  - Register to Gradescope
  - Consider attending recitation sessions
  - Start planning course project (topic, team, dataset)
    - Yes, really 😊
  - Complete Colab 0/1 released on Thursday
    - Colab 0/1 should each take you about one hour to complete (Note this is a “toy” homework to get you started. Real homeworks will be much more challenging and longer.)

- **Additional details/instructions at**
  http://www.cs.washington.edu/cse547
Distributed Computing for Data Mining
Large-scale Computing

- **Large-scale computing for data mining problems on commodity hardware**

- **Challenges:**

  - How do you distribute computation?
  - How can we make it easy to write distributed programs?
  - Machines fail:
    - One server may stay up 3 years (1,000 days)
    - If you have 1,000 servers, expect to lose 1/day
    - With 1M machines 1,000 machines fail every day!
An Idea and a Solution

- **Issue:**
  Copying data over a network takes time

- **Idea:**
  - Bring computation to data
  - Store files multiple times for reliability

- **Spark/Hadoop address these problems**
  - **Storage Infrastructure – File system**
    - Google: GFS. Hadoop: HDFS
  - **Programming model**
    - MapReduce
    - Spark
Storage Infrastructure

- **Problem:**
  - If nodes fail, how to store data persistently?

- **Answer:**
  - **Distributed File System**
    - Provides global file namespace

- **Typical usage pattern:**
  - Huge files (100s of GB to TB)
  - Data is rarely updated in place
  - Reads and appends are common
Distributed File System

- **Chunk servers**
  - File is split into contiguous chunks
  - Typically each chunk is 16-64MB
  - Each chunk replicated (usually 2x or 3x)
  - Try to keep replicas in different racks

- **Master node**
  - a.k.a. Name Node in Hadoop’s HDFS
  - Stores metadata about where files are stored
  - Might be replicated

- **Client library for file access**
  - Talks to master to find chunk servers
  - Connects directly to chunk servers to access data
Distributed File System

- Reliable distributed file system
- Data kept in “chunks” spread across machines
- Each chunk **replicated** on different machines
  - Seamless recovery from disk or machine failure

Bring computation directly to the data!

Chunk servers also serve as compute servers
Programming Model

- MapReduce is a **style of programming** designed for:
  1. Easy parallel programming
  2. Invisible management of hardware and software failures
  3. Easy management of very-large-scale data

- It has several **implementations**, including Hadoop, Spark (used in this class), Flink, and the original Google implementation just called “MapReduce”
MapReduce: Overview

3 steps of MapReduce

- **Map:**
  - Apply a user-written *Map function* to each input element
    - *Mapper* applies the Map function to a single element
      - Many mappers grouped in a *Map task* (the unit of parallelism)
    - The output of the Map function is a set of 0, 1, or more *key-value pairs*.

- **Group by key:** Sort and shuffle
  - System sorts all the key-value pairs by key, and outputs key-(list of values) pairs

- **Reduce:**
  - User-written *Reduce function* is applied to each key-(list of values)

Structure stays the same, **Map** and **Reduce** change to fit the problem
Map-Reduce: A diagram

**MAP:**
Read input and produces a set of key-value pairs

**Group by key:**
Collect all pairs with same key (Hash merge, Shuffle, Sort, Partition)

**Reduce:**
Collect all values belonging to the key and output
Map-Reduce: In Parallel

All phases are distributed with many tasks doing the work
MapReduce Pattern

![MapReduce Diagram]

Mappers

Reducers

key-value pairs

key-value pairs

Input

Output

Input

Output

Mappers

Reducers

Mappers

Reducers

01/02/2023
Example: Word Counting

Example MapReduce task:
- We have a huge text document
- Count the number of times each distinct word appears in the file

Many applications of this:
- Analyze web server logs to find popular URLs
- Statistical machine translation:
  - Need to count number of times every 5-word sequence occurs in a large corpus of documents
The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need.........................

The crew of the space shuttle Endeavor recently returned to Earth as ambassadors, harbingers of a new era of space exploration. Scientists at NASA are saying that the recent assembly of the Dextre bot is the first step in a long-term space-based man/machine partnership. "The work we're doing now -- the robotics we're doing -- is what we're going to need.........................
Word Count Using MapReduce

```python
map(key, value):
    # key: document name; value: text of the document
    for each word w in value:
        emit(w, 1)

reduce(key, values):
    # key: a word; value: an iterator over counts
    result = 0
    for each count v in values:
        result += v
    emit(key, result)
```
MapReduce: Environment

MapReduce environment takes care of:

- **Partitioning** the input data
- **Scheduling** the program’s execution across a set of machines
- Performing the **group by key** step
  - In practice this is the bottleneck
- Handling machine **failures**
- Managing required inter-machine **communication**
Dealing with Failures

- **Map worker (machine) failure**
  - Map tasks completed or in-progress at worker are reset to idle and rescheduled
  - Reduce workers are notified when map task is rescheduled on another worker

- **Reduce worker failure**
  - Only in-progress tasks are reset to idle and the reduce task is restarted
Spark
Problems with MapReduce

- **Two major limitations of MapReduce:**
  - Difficulty of programming directly in MR
    - Many problems aren’t easily described as map-reduce
  - Performance bottlenecks, or batch not fitting the use cases
    - Persistence to disk typically slower than in-memory work

- **In short, MR doesn’t compose well for large applications**
  - Many times one needs to chain multiple map-reduce steps
Data-Flow Systems

- MapReduce uses two “ranks” of tasks:
  One for Map the second for Reduce
  - Data flows from the first rank to the second

- Data-Flow Systems generalize this in two ways:
  1. Allow any number of tasks/ranks
  2. Allow functions other than Map and Reduce
     - As long as data flow is in one direction only, we can have the *blocking property* and allow recovery of tasks rather than whole jobs
Spark: Most Popular Data-Flow System

- Expressive computing system, not limited to the map-reduce model

- Additions to MapReduce model:
  - Fast data sharing
    - Avoids saving intermediate results to disk
    - Caches data for repetitive queries (e.g. for machine learning)
  - General execution graphs (DAGs)
  - Richer functions than just map and reduce

- Compatible with Hadoop
Spark: Overview

- Open source software (Apache Foundation)
- Supports **Java, Scala and Python**

- **Key construct/idea:** Resilient Distributed Dataset (RDD)
  - More on next slide.

- **Higher-level APIs:** DataFrames & DataSets
  - Introduced in more recent versions of Spark
  - Different APIs for aggregate data, which allowed to introduce SQL support
Spark: RDD

Key concept *Resilient Distributed Dataset* (RDD)

- Partitioned collection of records
  - Generalizes (key-value) pairs
- Spread across the cluster, Read-only
- Caching dataset in memory
  - Different storage levels available
  - Fallback to disk possible

- RDDs can be created from Hadoop, or by transforming other RDDs (you can stack RDDs)

- RDDs are best suited for applications that apply the same operation to all elements of a dataset
Spark RDD Operations

- **Transformations** build RDDs through deterministic operations on other RDDs:
  - Transformations include `map`, `filter`, `join`, `union`, `intersection`, `distinct`
  - **Lazy evaluation**: Nothing computed until an action requires it

- **Actions** to return value or export data
  - Actions include `count`, `collect`, `reduce`, `save`
  - Actions can be applied to RDDs; actions force calculations and return values
Task Scheduler: General DAGs

- Supports general task graphs
- Pipelines functions where possible
- Cache-aware data reuse & locality
- Partitioning-aware to avoid shuffles

Diagram:
- Stage 1: A: groupBy
- Stage 2: C: map, D: filter
- Stage 3: B: E: join, F: = RDD, = cached partition

Useful Libraries for Spark

- Spark SQL
- Spark Streaming – *stream processing of live datastreams*
- MLlib – *scalable machine learning*
- GraphX – *graph manipulation*
  - extends Spark RDD with Graph abstraction: a directed multigraph with properties attached to each vertex and edge
Data Analytics Software Stack

Spark Streaming
Stream processing

GraphX
Graph computation

MLlib
User-friendly machine learning

SparkSQL
SQL API

Spark
Fast memory-optimized execution engine (Python/Java/Scala APIs)

Tachyon
Distributed Memory-Centric Storage System

Hadoop Distributed File System (HDFS)

Mesos
Cluster resource manager, multi-tenancy

Hive

Storm

MPI

Hadoop MR
Spark vs. Hadoop MapReduce

- Performance: **Spark normally faster but with caveats**
  - Spark can process data in-memory; Hadoop MapReduce persists back to the disk after a map or reduce action
  - Spark generally outperforms MapReduce, but it **often needs lots of memory to perform well**; if there are other resource-demanding services or can’t fit in memory, Spark degrades
  - MapReduce easily runs alongside other services with minor performance differences, & works well with the 1-pass jobs (1 map, 1 reduce) it was designed for
- Ease of use: **Spark is easier to program** (higher-level APIs)
- Data processing: **Spark is more general**
Get course handout on website!

Recitation sessions:
- Spark tutorial using Colab 0:
  - Thursday, Jan 5, 3:30-5pm CSE2 371
- Review of basic probability and proof techniques
  - Tuesday, Jan 10, 3:30-5pm CSE2 371
- Review of linear algebra:
  - Thursday, Jan 12, 3:30-5pm CSE2 371