CS585 Final

Spring term, 2019-05-02
Duration: 1 hour

Instructions/notes
the exam is **closed** books/notes/devices/neighbors, and **open** mind :)

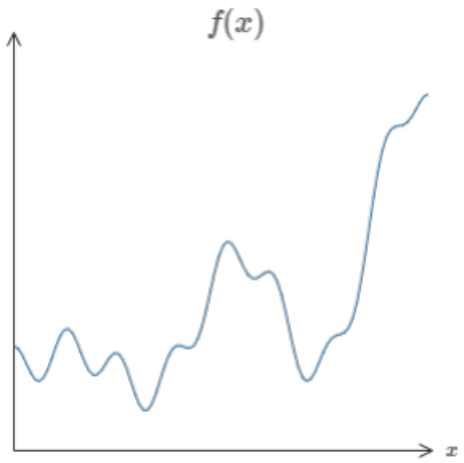
- there are 10 questions, together they touch upon every lecture topic [look for the word 'data' in each!], worth a total of 35 points
- the questions will make you think [not tax your memory]
- there are no 'trick' questions, or ones with long calculations or formulae
- there is a single blank 'scratch') page at the end, you can use it if you like
- **please do NOT cheat; you get a 0 if you are found to have cheated**
- **when time is up, stop your work**; you get a 0 if you continue

Good luck, have fun, hope you do well.

Q1 (3 points). 'Big Data Mining' involves making sense of ever-growing volumes of **data**. Also growing is the proportion of non-technical users (managers, domain experts, laypeople...) who want to analyze/mine all this Big Data. How would you, as a data expert (DBA + data scientist + data engineer...) ENABLE this, ie. what would you build (or buy) for their use? Note that you'd be placing yourself out of a job when you do this... :) Please be rather specific/technical in your description, instead of being vague.

A. The non-technical user would be best served by a 'data science platform' (appliance/turnkey solution), where they simply need to feed it input (eg. connect to a database, or streaming data source, etc), and do analysis via point-and-click, including creating dashboards for results. The idea is to not require an expert be available always, needing them only for occasional tuning, upgrading, training, etc. Examples of such platforms include Qubole, Splunk, Azure Databricks, Netezza, Dataiku.

Q2 (4 points). Given any function (that represents collected, labeled **data**, in any number of dimensions), including the 'wiggly' 2D one shown below, a neural network (NN) can always be constructed to approximate it (ie. to 'learn' the pattern in the labeled data). In this sense, an NN is a 'universal function approximator', which is the reason for its runaway success.



What, in math, is this (function approximation) roughly analogous to? Describe, using diagrams or simple equations.

A. This is loosely analagous to Fourier synthesis/summation/decomposition, where a periodic signal can be reconstructed using sines and cosines, eg:

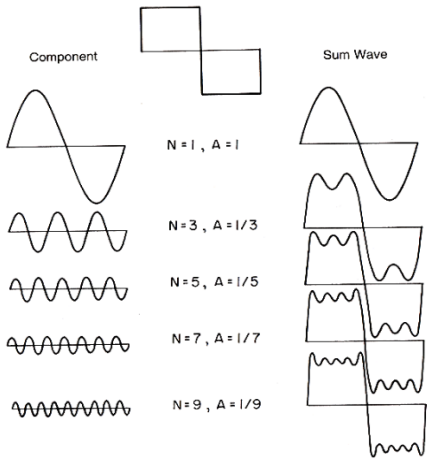
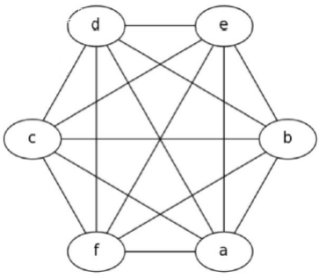


Figure 4–5 Fourier synthesis of a square wave. At the left are the successive harmonics; at the right are the sum waves including each successive harmonic. The graph at the top is the wave being synthesized.

From Berg and Stork

Q3 (2+2=4 points). Consider the following graph:



a. Represent the above, using a non-JSON, non-XML **data** format.

A.

Here is one way (verts, followed by edges):

a
b
c
d
e
f
a-b
b-e
e-d
d-c
c-f
f-a
a-e
e-c
c-a
f-d
d-b
b-f
a-d
e-f
b-c

b. What is the problem with using such formats (that aren't JSON or XML) for data description?

A. Such formats are difficult to parse, or at the least, would need a custom parser [prone to error, might lack features, might need resources to be developed, might need to be maintained ongoing...]. Further, adding new fields (eg. edge weights and labels, in the above example) would necessitate rewriting the parser, and might make the older data files unusable.

Q4 (1*3=3 points). 'BI' of yesteryear, looked a lot like this (PowerPoint slides, of a company's yearly sales **data**, presented at an annual meeting, for example):

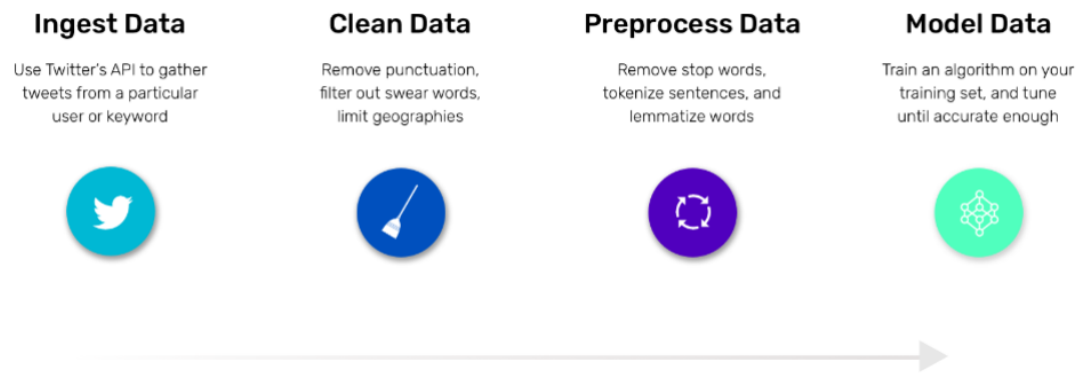
Todays BI is a quantum leap compared to the above - name/describe briefly, three ways in which it is much better today.

A.

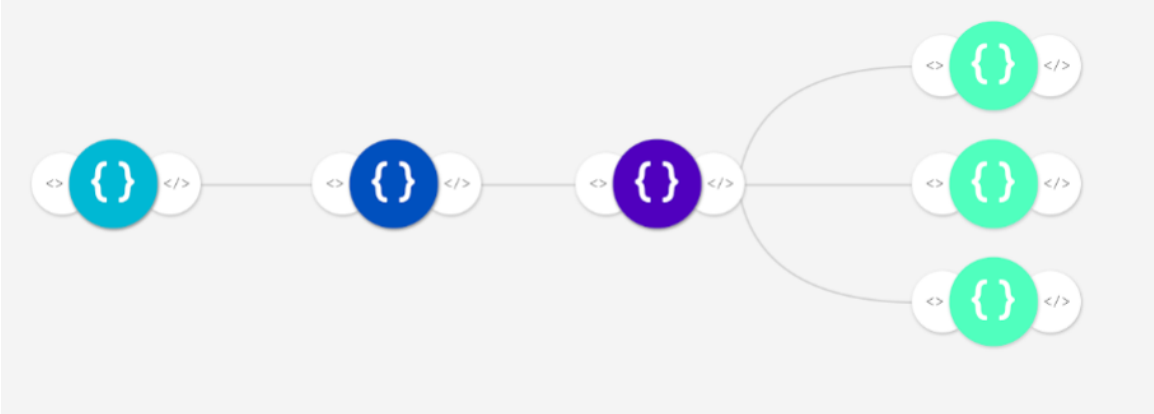
1. Interactivity, animation
2. Real-time (or near-real-time) processing, ie continuous analytics
3. Cloud processing
4. Mobile dashboards

...

Q5 (2+2=4 points). Here is a standard way to analyze Twitter feed **data** (eg. you could create it via a Jupyter notebook, using Python calls to an API or APIs):



Here is a better way:



In the 'better' way above, the processing pipeline is expressed as a graph, where each {} represents a node, inside which the functionality (ingesting data, cleaning...) could be expressed via a function or microservice call. What is such an analysis scheme called, and why is it a better alternative?

A. This would be 'dataflow', which is better because:
* an individual step [ie a node's contents/calls] could be swapped out, possibly to improve performance, reduce operating costs...
* changes made to one node will only require recomputing downstream nodes (as opposed to having to re-execute the entire pipeline)
* nodes could be run in parallel where possible
* nodes could be distributed across devices, cloud...

Mentioning two of the four items above, would be a sufficient answer.

Q6 (1*3=3 points). In a small company, a classic **data** science usage scenario would look like this: a team of data scientists comes up with, or selects, a data mining algorithm for use, working with domain experts to learn about the data to analyze; a team of data engineers would then build a pipeline around this algorithm (including coding the algorithm from scratch, or using API calls for it from a library), test, and deploy it on the company's servers, along with the data, for mining/learning and ongoing usage.

What are three current/emerging trends that are alternatives to the above?

A.

1. Using the cloud.
2. Using VMs or containers [VirtualBox, Docker, Kubernetes...]
3. Using a platform.
...

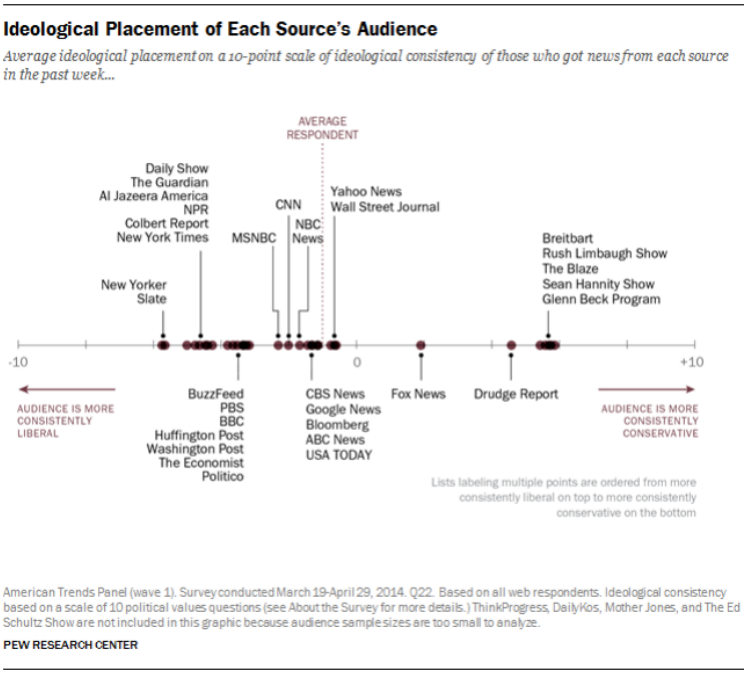
Q7 (1+3=4 points). As you know, very large graph datasets (eg. from social media, web page linking, etc) are commonplace. 'Connected components' (clusters of inter-connected vertices) is an algorithm that is run on such graph **data** [eg. to divide a set of web pages into groups (clusters), where in a group, pages link to each other, but not to any page outside the group]. MapReduce can be used for this computation, after partitioning the data into horizontal fragments.

However, use of MapReduce would require looping through several map-reduce iterations, to grow the components/clusters incrementally. What is a better alternative, and why?

A. BSP is a better algorithm for this, because nodes can process data (expand a vertex's neighborhood incrementally) till they need to exchange data with other nodes (eg. pairs of nodes that contain an edge that is split up because of partitioning) in order to complete the partitioning.

Note that 'YARN' is not the best answer, because it is simply an augmentation of the original (v1) MapReduce algorithm [MapReduce is already mentioned in the question].

Q8 (1.5+1.5=3 points). The following 'political **data**' visualization shows where, in the left/right political "spectrum", various audiences that consume content from news outlets lie (we went over this in class) - left is liberal, right is conservative:



What are a couple of ways by which you could make the above depiction more informative (ie. how would you improve it), if you could gather more data?

A.

1. We could create a 'bubble' chart, with readership/viewership info - bigger bubbles mean a bigger audience.

2. We could indicate, using a bar chart for example, how many news items each outlet publishes, each day/month/year...

...

Q9 (2+2=4 points). People increasingly have 'always on' microphones in their homes, in the form of Alexa/Echo, Google Assistant, etc. “Alexa” at times responds when “she” is not spoken to (when this happens, your speech not intended for Alexa does get transmitted to Amazon's servers). Also, it has been reported that a ('small') team of Amazon ML engineers are authorized to listen in on anonymized conversations at random, with the goal of making interactions better – but, they do have access to at-home devices' location coordinates (which means they might be to figure out an 'anonymous' user's location).

What can go wrong, from a privacy and security POV, when (a) voice **data** not meant for Alexa gets sent to Amazon's servers, and (b) a malicious employee might be able to locate a user who is supposed to have been anonymous? Be as specific as possible.

A. If Alexa (eg. an Echo device at home) transmits a user’s speech incorrectly (ie. data not meant to be consumed by Alexa) to Amazon’s servers, that could lead to the user be profiled (info can be learned about them), and consequently blackmailed; a government subpoena of the user’s Alexa queries would lead to all this extra data be sent to the government, with all sorts of repurcussions [extra surveillance, imprisonment etc, depending on the content of those extra conversations]. If a malicious employee is able to locate a user’s address, that employee can publish it online, sell the info, notify local police about supposedly ‘bad’ intent on the user’s part, etc. The bottom line is that a typical user is unaware that their private thoughts and location might be available to people whom they will never know/meet, and has no idea how, or when, or for what purpose, such info might be misused.

Q10 (3 points). Unlike the dominating AI algorithms today (esp. neural networks, including DL, CNN etc), 'AGI' (artificial general intelligence, ie. “next gen” AI) is not expected to be data-hungry at all. Why would an AGI system not need to rely heavily on training **data** (ie. how could it be constructed)? You can explain in your own words, based on what was discussed in class, or based on your own ideas.

A. An AGI system would be expected to learn **concepts/features** related to the world (everything from physical phenomena (eg. rain), bodily and mental feelings, human behavior, etc) by experiencing the world directly, and reason/act using them. Such concepts become categorized, and generalized, and interlinked with other concepts, forming a giant knowledge graph of sorts. For all this, we don’t need large amounts of training data [in the real world, humans and animals, don’t]. In contrast, a standard DL network, lacking any features, solely relies on input, to learn to categorize - that is the reason it needs excessive amounts of data.