Applied Mathematics 120:
Applied Linear Algebra and Big Data

Extended syllabus (most recent version available here)

FAS course web page: https://canvas.harvard.edu/courses/46905 *(Spring 2019)*

Last updated: Tuesday 28th May, 2019, 15:08.

1 Administrative

Instructor: Eli Tziperman (eli@seas.harvard.edu); **TFs:** Please see course web page. Feel free to email or visit us with any questions.

Day, time & location: Tue, Thu 10:30-11:45, Jefferson 250.

Office hours: Each of the teaching stuff will hold weekly office hours, see course web page for times & place. Eli's office: 24 Oxford, museum building, 4th floor, room 456.

Course resources: (1) The **detailed syllabus** lists material used for each lecture. (2) The **course notes**. (3) The **supplementary materials**, including Matlab/ python demos. Require VPN from the FAS download site to access from outside campus & from the Harvard wireless network.

Prerequisites: Applied Mathematics 21a and 21b, or equivalent; CS50 or equivalent.

Computer Skills: Programming experience is expected, homework assignments involve significant code writing, you may use either Matlab or python.

Using Matlab: install from FAS software downloads. For a Matlab refresher, register in advance to the Matlab boot camp, 3 lectures at the start of the term.

Using python: Install anaconda for python 3.7. Course demos have been tested using the "spyder" interface. You may want to consider Jupyter notebooks.

Sections/ weekly HW help sessions: Monday 5–7pm, or as advertised, the EPS faculty lounge on 4th floor of Hoffman, 20 Oxford St. Please come to work on the homework assignments, ask questions, and to offer help to others.

Homework will be assigned every Tuesday under "Files" on Canvas, due (via electronic submission, see below) the following Tuesday at 10:00am, unless otherwise noted. Continuously practicing the lecture material on a weekly basis via HW assignments is the only way to become comfortable with the subjects covered in the course.

- **Course forum:** Please post questions regarding HW/ quizzes to the course forums (piazza.com/harvard/spring2019/apmth120), rather than emailing the teaching staff. You are very welcome to respond to other students' questions on the forum.
- **Electronic homework submission via Canvas:** Your submission, including code and figures, must be a single PDF file, no more than 20 Mb in size and no more than 30 pages in total length. It may be typeset or scanned using scanners available in the libraries, but must be clear, easily legible, and correctly rotated. A scan using

a phone app (e.g., this) may be acceptable if done carefully. Unacceptable scans could lead to a rejection of the submission or to a grade reduction of 15%. Late submissions would lead to a reduction of 2% per minute after the due time.

Quizzes, final, grading: Homework: 40%; two quizzes, (tentatively) scheduled to

1. Wednesday, March 6, 2019, 7-9pm, Geological Museum lecture hall 100, 24 Oxford St.
2. Wednesday, April 10, 2019, 7-9pm, Yenching Auditorium, room 118, 2 Divinity Ave.

(all in the evening): 30% together; final: 30%. Your lowest HW grade will be ignored. HW and quiz grades are posted to canvas, you need to check the posted grades and let us know within 7 days from the release of grades if you see a problem; later responses cannot be considered. Please approach Eli rather than the TFs with any issue related to grading.

Collaboration policy: we strongly encourage you to discuss and work on homework problems with other students and with the teaching staff. However, after discussions with peers, you need to work through the problems yourself and ensure that any answers you submit for evaluation are the result of your own efforts, reflect your own understanding and are written in your own words. In the case of assignments requiring programming, you need to write and use your own code, code sharing is not allowed. You must appropriately cite any books, articles, websites, lectures, etc that have helped you with your work.

Textbooks:

Str: Strang, G, *Linear Algebra and its Applications, 4th ed., 2006* (see also this&this)

MMD: Leskovec, Rajaraman and Ullman, *Mining of Massive Datasets*, download,

Nielsen: Michael Nielsen, online book "Neural networks and deep learning".

Contents

2 Outline

Topics in linear algebra which arise frequently in applications, especially in the analysis of large data sets: linear equations, eigenvalue problems, linear differential equations, principal component analysis, singular value decomposition, data mining methods including frequent pattern analysis, clustering, outlier detection, classification, and machine learning, including neural networks and random forests. Examples will be given from physical sciences, biology, climate, commerce, internet, image processing and more.

Please see here for a presentation with a review of example applications.

3 Syllabus

Follow links to see the source material and Matlab/python demo programs used for each lecture under the appropriate section of the course downloads web page.

1. Introduction, overview. sources.
   We'll discuss some logistics, the course requirements, textbooks, overview of the course, what to expect and what not to expect (presentation).

2. Linear equations. Sources. Notes: chapter 2.
   (a) Notation
   (b) Motivation: matrices and linear equations arise in the analysis of electrical network, chemical reactions, large ones arise in network analysis, Leontief economic models, ranking of sports teams (**Str**§2.5; also **Str**§8.4), numerical finite difference solution of PDEs, and more.

(c) Medical tomography as an example application of linear equations, which may lead to either under or over determined systems (notes, section 2.1.1).

(d) Reminder: row and column geometric interpretations for linear equations $A\mathbf{x} = \mathbf{b}$, $a_{ij}x_j = b_i$ (notes §2.2; **Str**§1.2, 2d example; `geometric_interpretation_of_linear_eqns_in_3d.m/py`). Solving linear equations using Gaussian elimination and back substitution (**Str**§1.3). Cost (number of operations, **Str**§1.3).

(e) Solution of large linear systems via direct vs iterative techniques

i. Direct method: LU factorization (notes §2.3; **Str**§1.5, see Matlab/python demos of both a detailed hand-calculation for a 3x3 matrix and a using library routines here. (Optional: More on why the LU decomposition algorithm works here, or in chapters 20, 21 of Trefethen and Bau III (1997)).

ii. Iterative methods: Jacobi, Gauss-Seidel, (Time permitting: SOR) (notes §2.4; **Str**§7.4; a code with an `SOR_example.m/py`, and SOR derivation notes; convergence is further discussed in notes by RAPETTI-GABELLINI Francesca, and typically systems based on matrices that are either diagonally-dominant, or symmetric positive definite, or both, tend to converge best).

(f) Does a solution exist and is it sensitive to noise/ round-off error? (Notes §2.5, **Str**§1.7). Two examples from showing the effects of ill conditioned matrix and of using wrong algorithm even with a well conditioned matrix.

(g) Dealing with huge systems:

i. Special cases: sparse, banded and diagonal matrices (notes §2.6.1; wikipedia and `sparse_matrix_example.m/py`) [HW: solving tridiagonal systems]. Bad news: LU factorization of a sparse matrix is not necessarily sparse (Figure and a an example, `LU_of_sparse_matrix.m/py`), so it might be best to use an iterative method to solve the corresponding linear system of eqns.

ii. Google's MapReduce (Hadoop) algorithm: general idea (notes §2.6.2; **MMD**§2 intro, pp 21-22). Examples: calculating mean daily flight delay (code: `MapReduce_Mean_Daily_Flight_Delay_example.m/py`) and corresponding output file; matrix-matrix multiplication using one MapReduce step (**MMD**§2.3.10 pp 39-40, video and text links); (Time permitting:) the more efficient two step approach (**MMD**§2.3.9).

3. Eigenvalues, eigenvectors. Sources. Notes: chapter 3.

(a) Motivation: Google's PageRank; partitioning (clustering) of graphs/ networks; differential equations (**Str**§5.1) and explosive development of weather systems.

(b) Reminder: Eigenvalue problems $A\mathbf{x} = \lambda\mathbf{x}$, finding eigenvalues through $\det(A - \lambda I) = 0$, then finding eigenvectors by solving $(A - \lambda_i I)\vec{e}_i = 0$ (**Str**§5.1).

Similarity transformation $S^{-1}AS$ and diagonalization of matrices with a full set of eigenvectors (**Str**§5.2) and of symmetric matrices (**Str**§5.6, 5S).

(c) Google's PageRank algorithm: (notes §3.2) First, Google vs BMW: here. Modeling the Internet via a random walker and the PageRank algorithm from p 1-7 here. See demo codes. It turns out that PageRank is the eigenvector with the largest eigenvalue of the transition matrix. The theoretical background, proving that there is a PageRank and that it is unique is the Perron-Frobenius theorem stating that a stochastic matrix (each row sums to one) with all positive elements has a single largest eigenvalue equal to one. See Wikipedia for the theorem and for stochastic matrices;

(d) The power method (notes §3.3)

i. Calculating the largest eigenvalue/ vector;

ii. Reminder: orthonormal base; orthogonality and projection of vectors (projection of $\vec{b}$ in the direction of $\vec{a}$ is $(\vec{b}\cdot\vec{i}_a)\vec{i}_a = (|b|\cos\theta)\vec{i}_a$ using the unit vector $\vec{i}_a = \vec{a}/|a|$); Gram-Schmidt orthogonalization (**Str**§3.4).

iii. Calculating the largest p eigenvalues/ vectors using the block power method

iv. The inverse power method for calculating the smallest eigenvalue/ eigenvector;

v. The more efficient shifted inverse power method (**Str**§7.3; example code: `block_power_method_example.m/py`; it seems that the block method should work only for normal matrices, whose eigenvectors are orthogonal, although Strang does not mention this);

(e) Spectral clustering (partitioning) of networks via eigenvectors of corresponding Laplacian matrices (notes §3.4)

i. Preliminaries: More on networks and matrices: Transition matrix was covered already as part of the PageRank algorithm above (**MMD** example 5.1, p 166). Adjacency matrix (example 10.16, p 363), Degree matrix (example 10.17, p 364), Laplacian matrix (example 10.18, p 364).

ii. Spectral clustering (code, `network_classification_example.m/py` and notes, expanding on **MMD**§10.4.4 and example 10.19, pp 364-367).

(f) (Time permitting:) Solving large eigenvalue problems efficiently: QR (Gram-Schmidt) factorization and Householder transformations (**Str**§7.3)

(g) Generalized eigenvalue problems (notes §3.5) $A\mathbf{x} = \lambda B\mathbf{x}$, arise in both differential equations and in classification problems (see later in the course). If A, B are symmetric, it is not a good idea to multiply B^{-1} to obtain a standard eigenproblem because $B^{-1}A$ is not necessarily symmetric. Instead, transform to a regular eigenvalue problem using Cholesky decomposition (code, `Generalized_eigenvalue_problem.m/py`, and notes).

(h) Linear ordinary differential equations and matrix exponentiation (notes §3.6; **Str**§5.4, remark on higher order linear eqns, heat PDE example on). Eigenvalues and stability (**Str**§5.4; phase space plots and Romeo and Juliet from Strogatz (1994) §5.3). Matlab/python demos: first run love_affairs(1) and then run_all_ODE_examples.m/py. Emphasize that solution behavior is determined by real and imaginary part of eigenvalues.

(i) Dramatic surprises on the path to tranquility: Non-normal matrices, transient amplification and optimal initial conditions (notes §3.7, and code, `non-normal_transient_amplification.m/py`).

(j) Jordan form and generalized eigenvectors: when straightforward diagonalization using standard eigenvectors doesn't work because they are not independent (notes §3.8)

i. Start with simple example of the issue using the beginning of the following code, `Jordan_demo.m/py`.

ii. Definition and statement of the ability to always transform to Jordan normal form (**Str**, 5U).

iii. Second order ODE equivalent to a first order set in Jordan form, that leads to a resonant solution, see notes.

iv. How to find the Jordan form using the matrix of generalized eigenvalues detailed example of a simple case. (Time permitting: additional details in **Str** App B; and in notes on the more general case by Enrico Arbarello).

v. Extreme sensitivity to round-off error: demonstrated by final part of above Matlab/python demo.

vi. (Time permitting:) Proof by recursion that a Jordan form can always be found is also in **Str** Appendix B.

4. Principal component analysis, Singular Value Decomposition. Sources. Notes: chapter 4.

(a) Motivation: dimension reduction, e.g., image compression, face recognition, El Niño; comparing structure of folded proteins; more unknowns than equations

(b) Principal Component Analysis (PCA; also known as Factor Analysis or Empirical Orthogonal Functions), calculation from covariance matrix (notes §4.1).

(c) Singular Value Decomposition (SVD): statement, examples, and practical hints for calculating the SVD decomposition, $X = U\Sigma V^T$ (notes §4.2.1; **Str**§6.3 including remarks 1,2,4,5 and examples 1,2; note that $A\mathbf{u}_i = \sigma_i \mathbf{v}_i$ and $A^T\mathbf{v}_i = \sigma_i \mathbf{u}_i$ these are therefore "right and left eigenvectors"). Note: eigenvectors of a symmetric matrix $A = A^T$ are orthogonal because this matrix is also normal, see proof here.

(d) Geometric interpretation of SVD for the special case of a real square matrix with a positive determinant (notes §4.2.2; see animation and caption from Wikipedia by Kieff, with some more details here.

(e) SVD applications (notes §4.3)

i. Image compression, low-rank approximation, (notes §4.3.1; **Str**§6.3, code: SVD_applications_image_compression.m/py), variance explained.

ii. Effective rank of a matrix (notes §4.3.2; **Str**§6.3, matrix condition number and norm (**Str**§7.2). Code, SVD_applications_matrix_rank_norm_condition_number.m/py).

iii. Polar decomposition (notes §4.3.3; **Str**§6.3). Applications exist in computer animation (notes §4.3.3.1), comparing protein structures (notes §4.3.3.2), continuum mechanics, robotics, and more.

A. A simple demo, SVD_applications_polar_decomposition_example.m/py, of the geometric interpretation of polar decomposition.

B. Computer animation (notes §4.3.3.1; SVD_applications_polar_decomposition_animation.m/py)

C. The polar-decomposition-based Kabsch Algorithm for comparing protein structures using the root-mean-square deviation method (notes §4.3.3.2, (Kavraki, 2007, p 1-5), and a demo, SVD_applications_polar_decomposition_Kabsch_example.m/py.

D. (Time permitting:) proof that polar decomposition of the correlation matrix between molecule coordinates is indeed the optimal rotation matrix (notes §4.3.3.2, following).

iv. When number of unknowns is different from number of equations:

A. Overdetermined systems: more equations than unknown and least squares. (i) Brief reminder (notes §4.3.4). (ii) Using QR decomposition (cover it first if it was not covered in the eigenvalue/ vector section) for an efficient solution of least-square problems (**Str**§7.3).

B. Under-determined systems, more unknowns then equations: pseudo inverse solution using SVD and a short proof that it is indeed the smallest-norm solution (notes §4.3.5; **Str**§6.3; example, SVD_application_underdetermined_linear_eqns.m/py).

C. rank deficient over-determined or under-determined systems: $r < \min(N, M)$ (notes § 4.3.6).

D. A review of all types of linear equations using the code Review_examples_linear_equations.m/py).

v. PCA using SVD: (notes § 4.3.7 or Hartmann; and an example, PCA_small_data_example_using_SVD.m/py for PCA using SVD.

vi. Multivariate Principal Component Analysis and Maximum Covariance Analysis (MCA): analysis of two co-varying data sets. E.g., M stocks from NY and L stocks from Tokyo, both given for N times: Y_{mn}, T_{ln} (notes §4.3.8; Matlab/python demos in Sources and links from these notes).

vii. The Netflix challenge part I: latent factor models and SVD. First, highlighted text and Figs. 1 and 2 on pp 43-44 of Koren et al. (2009); then, notes §4.3.10; finally, example code, `SVD_applications_Netflix.m/py`. [optional: information on the fuller procedure in highlighted parts of section 6.1 of Vozalis and Margaritis (2006) available here; Instead of eqn (4) in Vozalis, let the predicted rating of movie a by user j be $pr_{aj} = \sum_{i=1}^{n} sim_{ji}(rr_{ai} + \bar{r}_a)/(\sum_{i=1}^{n} |sim_{ji}|)$, where sim_{ji} is the similarity between the ratings of movies i, j by all users, and the sum is over movies)].

5. Similar items and frequent patterns. Sources. Notes: chapter 5.

(a) Motivation for similar items: face recognition, fingerprint recognition, comparing texts to find plagiarism, Netflix movie ratings. (**MMD**§3)

(b) Similar items:

i. Jaccard Similarity index (**MMD**§3.1.1 p 74; demo, `Jaccard_examples.m/py`, for logicals, numbers, text files).

ii. Converting text data to numbers: Shingles, k-shingles, hashing, sets of hashes (**MMD**§3.2 p 77-80; section 1 of notes and corresponding Matlab/python demo of an oversimplified hash function; another demo, `crc32_demo.m/py`, for the Cyclic Redundancy Check (crc32) hash function)

iii. Matrix representation of sets (**MMD**§3.3.1 p 81)

iv. (Time permitting:) MinHash algorithm for comparing sets (**MMD**§3.3 p 80-86, and section 2 of notes with summary of MinHash steps)

A. Minhashing: creating a similarity-conserving signature matrix that is much smaller than the original data matrix, and that allows for an efficient comparison of sets. Signature matrix is based on a set of random permutations of the rows of the data matrix (**MMD**§3.3.2,§3.3.4 p 81-83)

B. "Proof" that the probability of having similar MinHash signatures of two sets is equal to the Jaccard similarity of the two sets (**MMD**§3.3.3 p 82-83)

C. MinHash signature estimated using a set of random hash functions acting on the data matrix (**MMD**§3.3.5 p 83-86)

D. Additional resources: code, `MinHash_and_signature_matrix_example.m/py`, for calculating signature matrix and using it to estimate Jaccard similarity; A more elaborate example python code by Chris McCormick, run using `spyder`)

E. Locality-Sensitive Hashing (LSH, **MMD**§3.4-3.8)

(c) Motivation for frequent patterns: market basket analysis: hot dogs and mustard, diapers and beer; frequent combined Internet searches: Brad and Angelina; medical diagnosis: biomarkers in blood samples and diseases; detecting plagiarism. (**MMD**§6)

(d) Frequent patterns and association rules.

i. Mining frequent patterns (and association rule learning): support for set I (number of baskets for which I is a subset); I is frequent if its support is larger than some threshold support s; (**MMD**§6.1 p 201-206)

ii. Association rules $I \to j$ between a set I and an item j; confidence (fraction of baskets with I that also contain j) and interest (difference between confidence in $I \to j$ and fraction of baskets that contain j); (**MMD**§6.1.3-6.1.4)

iii. Apriori algorithm: (**MMD**§6.2, highlighted parts on p 209-217)

A. Baskets as sets of numbers (**MMD**§6.2.1 p 209)

B. Monotonicity of itemsets (**MMD**§6.2.3 p 212)

C. A-priory first pass; renumbering of relevant itemsets between passes; and second pass to identify frequent pairs (**MMD**§6.2.5; a simple code, `apriori_example.m/py`)

D. Beyond frequent pairs: larger frequent itemsets (**MMD**§6.2.6)

E. Example of finding association rules via A-priori algorithm, Matlab code by Narine Manukyan, run using `demoAssociationAnalysis;`

6. Unsupervised learning: cluster analysis. Sources. Notes: chapter 6.

(a) Motivation: *Archaeology/Anthropology:* group pottery samples in multiple sites according to original culture; *Genetics:* clustering gene expression data groups together genes of similar function, grouping known genes with novel ones reveals function of the novel genes; *TV marketing:* group TV shows into groups likely to be watched by people with similar purchasing habits; *Criminology:* clustering Italian provinces shows that crime is not necessarily linked to geographic location (north vs south Italy) as is sometimes believed; *Medical imaging:* measuring volumes of cerebrospinal fluid, white matter, and gray matter from magnetic resonance images (MRI) of the brain using clustering method for texture identification; *Internet/ social networks:* identify communities; *Internet search results:* show relevant related results to a given search beyond using keywords and link analysis; *Weather and climate:* identify consistently re-occurring weather regimes to increase predictability.

(b) Overview: Two main approaches to clustering: hierarchical (each point is an initial cluster, then clusters are being merged to form larger ones) and

point-assignment (starting with points that are cluster representatives, clusteroids, and then adding other points one by one). Other considerations: Euclidean vs non, and large vs small memory requirements (**MMD**§7.1.2, p 243).

(c) Distances/ metrics (notes §6.2)

i. Requirements from a distance: **MMD**§3.5.1, p92-93.

ii. Examples of distance functions (**MMD**§3.5, p 93-97): Euclidean (L_2 distance), L_r distance, Manhattan (sum of abs values, L_1 norm), maximum (L_∞), Hamming distance between two strings of equal length or between vectors of Booleans or other vectors, cosine (difference between angles), Jaccard distance (one minus Jaccard similarity), edit. Noting that "average" distance does not necessarily exist in non Euclidean spaces (p97).

(d) Curse of dimensionality: problems with Euclidean distance measures in high dimensions, where random vectors tend to be far from each other and perpendicular to each other, making clustering difficult (notes §6.3; **MMD**§7.1.3 p 244-245, code: `curse_of_dimensionality.m/py`)

(e) Hierarchical clustering (notes §6.4): intro and example (**MMD**§7.2.1, Figs 7.2, 7.3, 7.4, 7.5, and the resulting dendogram in Fig 7.6), efficiency (**MMD**§7.2.2, p 248-249), merging and stopping criteria (**MMD**§7.2.3), in non Euclidean spaces using clustroids (**MMD**§7.2.4, p 252-253). (Use `run_hierarchical_clustering_demos.m/py` to run three relevant demos: First detailed hand calculation, then the example script run first with argument (2) and then with (20). The `hierarchical_clustering_simpler_example.m/py` code there is a bare-bone version that can be useful in HW)

(f) K-means algorithms (notes §6.5): these are point-assignment/ centroid-based clustering methods.

i. Basics (**MMD**§7.3.1),

ii. Initialization (**MMD**§7.3.2; e.g., initialize centroids on k farthest neighbors)

iii. Choosing k (**MMD**§7.3.3).

iv. Demos: using `run_kmeans_clustering_demos.m/py`, first a detailed hand calculations and then the a more detailed example.

(g) Self-organizing maps ("Kohonen maps", a type of an artificial neural network; notes §6.6).

(h) Mahalanobis distance: first for stretched data, diagonal covariance matrix, then non-diagonal, stretched and rotated (notes §6.7).

(i) Spectral clustering into two or more sub-clusters (notes §6.8). Such clustering using eigenvector 2 was already covered for networks, using the Laplacian matrix of the network, in the eigenvalues/ eigenvectors section.

i. First a reminder of network clustering (notes §3.4).

ii. Then for clustering of other data: Form a distance matrix $s_{ij} = |\mathbf{x}_i - \mathbf{x}_j|$, defined here as the distance between points i and j in the set; then a "similarity" matrix (equivalent to adjacency matrix in network clustering) using, e.g., $w_{ij} = \exp(-s_{ij}^2/\sigma^2)$, then a diagonal degree matrix $d_i = \sum_j w_{ij}$, and finally the Laplacian matrix $L = D - W$ (highlighted parts in p 1-4 of Von-Luxburg (2007)

iii. A proof that the quadratic form $\mathbf{x}^T L\mathbf{x}$ is equal to the sum over squared differences of linked pairs (Proposition 1 on p 4 of Von-Luxburg, 2007)

iv. Demos of dividing data into two clusters, first two examples in run_spectral_clustering_examples.m/py.

v. Two options for dividing data into more than two clusters: (1) Wikipedia adds that "The algorithm can be used for hierarchical clustering by repeatedly partitioning the subsets in this fashion". (2) More interestingly, can also cluster into $k > 2$ parts using eigenvectors 2 to k, see box on section 4 on p 6 of Von-Luxburg (2007), and the third example in run_spectral_clustering_examples.m/py.

vi. Benefit: clustering n data vectors that are only $(k-1)$-dimensional [eigenvectors 2–k]; Much more efficient than clustering original n d-dimensional data points where d could be much larger than $(k-1)$.

(j) BFR algorithm: Clustering large data sets that cannot be fully contained in memory: BFR algorithm and Summarization (notes §6.9; **MMD**§7.3.4 and 7.3.5, p 257 to middle of 261)

(k) CURE (Clustering Using REpresentatives) algorithm, for clusters that have complex shapes, such as concentric rings. This is a point-assignment clustering algorithm, like k-means, not relying on centroids but on a set of representative points that span an entire complex-shaped cluster (notes §6.10; **MMD**§7.4, p 262-265; and several demos using `run_CURE_examples.m/py` of Hierarchical clustering based on an appropriate distance measure to find the representatives and then point assignment to cluster the rest of the data)

(l) (Time permitting:) Outlier/ anomaly detection: a brief overview only. Motivation: unusual credit activity as indication of credit card theft. Detection using statistical methods e.g., assuming Gaussian distribution;

7. Supervised learning: classification. Sources. Notes: chapter 7.
(We stick to Euclidean distances for now, other options were discussed under cluster analysis).

(a) Motivation: Optical character recognition, handwriting recognition, speech recognition, spam filtering, language identification, sentiment analysis of tweets (e.g., angry/ sad/ happy), amazon book recommendation, Netflix challenge, on-line advertising and ad blocking on Internet sites, credit scores, predicting loan defaulting, and Mastering the game of Go!

(b) Machine learning Introduction (MMD§12.1, p 439-443)

(c) Perceptrons: Intro (notes §7.2; **MMD**§12.2 p 447); zero threshold (**MMD**§12.2 first two paragraphs, 12.2.1, p 448-450); allowing threshold to vary (**MMD**§12.2.4, p 453); problems (**MMD**§12.2.7, simply show Figs. 12.11,12.12,12.13 on p 457-459). Use `perceptron_classification_example.m/py`, see comments at top of code for useful cases to show; for adjustable step I made step size (η) proportional to deviation of current data point that's not classified correctly ($\eta = |\mathbf{x} \cdot \mathbf{w} - \theta|$), but bounded on both sides, say $0.01 < \eta < 1$.

(d) Support vector machines (notes §7.3; **MMD**§12.3, p 461-469);

i. Introduction, formulation for separated data sets, formulation for overlapping data sets, solution via gradient method and a numerical Example 12.9 of the final algorithm

ii. Misc Matlab/python demos, run all relevant ones using `run_SVM_demos.m/py`.

(e) A brief introduction to Multi-Layer Artificial "feed forward" Neural Networks (notes §7.4):

i. Motivation: these are based on a powerful extension of the perceptron idea, and allow computers to perform image/ voice/ handwriting/ face recognition, as well as Mastering the game of Go.

ii. Introduction: perceptron as a neural network with no hidden layers; failure of perceptron for XOR, and success using one hidden layer and a simple nonlinear activation function; a general one-hidden layer formulation (highlighted parts of the introductory notes by Lee Jacobson)

iii. Details: architecture (including number of layers, number of nodes in each layer, geometry of connections between nodes); example activation functions: tansig, sigmoid, rectified linear, softplus; selecting output layer activation function based on need for (1) regression (linear output layer), (2) a yes or no (sigmoid output layer), (3) a discrete set of labels using a softmax output layer plus Matlab's vec2ind (on-line demo), (Goodfellow et al. (2016), §6.2.2, p 181-187; the activation functions are plotted by `neural_networks0_activation_functions_examples.m/py` and in Goodfellow et al. (2016), §3.10, and Figs. 3.3, 3.4, p 69)

iv. Matlab/python demos, use run_neural_network_demos.m/py to run all, stop just before backpropagation demos which are shown later.

A. Understanding the internals of Matlab/python's neural networks using `neural_networks1_reverse_engineer_manually.m/py`.

B. Two simple example neural network Matlab/python example codes for classification, `neural_networks2_simple_2d_classification_example.m/py` and

regression, neural_networks3_simple_2d_regression_example.m/py, and then a failed network, neural_networks4_simple_2d_FAILED_classification_example.m/py, to show how this can be diagnosed.

C. An example, neural_networks5_character_recognition_example_appcr1_Mathworks.m/py, that demonstrates character recognition using Matlab's neural network toolbox.

v. Back-propagation! (notes §7.4.3) Calculating the cost gradient with respect to weights and biases (**Nielsen**)

A. Cost function definition (**Nielsen**§1, eqn 6)

B. Gradient descent rule (**Nielsen**§1, eqns 16,17, and following two paragraphs).

C. Back-propagation: basically all of **Nielsen**§2.

D. Code demos: continue running run_neural_network_demos.m/py from where we stopped previously, which will show the following. First, hand calculation demonstrated in neural_networks6_backpropagaion_hand_calculation.m/py, of feedforward and backpropagation, comparing to a finite-difference estimate of gradient. Then, a full optimization of a neural network, neural_networks7_backpropagation_and_steepest_descent.m/py, first with MNIST=0 for a simple XOR data set, and then with 1, for an actual hand-writing recognition data set (translated to Matlab from a python code by **Nielsen**)

vi. Ways to recognize problems and choose parameters for neural networks (notes §7.4.4)

A. Learning slow-down and the improved *cross-entropy cost function* that resolves that (appropriate section of **Nielsen**§3, beginning to two demos after eqn 62. Use on-line version of the chapter for the nice demos.)

B. Over-fitting, how to identify it and how to resolve it using (1) L2 regularization and (2) enlarging the training data set using random rotations/ added noise to original data (appropriate section of **Nielsen**§3)

C. Weight initialization to avoid initial saturation and increase initial learning rate (appropriate section of **Nielsen**§3)

D. Choosing network's hyper-parameters: learning rate (which may also vary with epochs), regularization constant, mini-batch size used the average the gradient before applying steepest descent. Trick is to first find parameters that lead to *any* learning, and improve from there (appropriate section of **Nielsen**§3)

E. Convolution layers and their advantages: parameter sharing, sparse interactions (Goodfellow et al. (2016), §9.1-9.2); zero-padding in convolution (Fig 9.13, p 351); pooling (§9.3);

(f) k-nearest neighbors (k-NN) (notes §7.5)

i. Classification: finding a label of input data based on majority of k nearest training data neighbor(s) when label is discrete such as type of dog or sick vs healthy. Start with a single neighbor, including Voronoi diagram (**MMD**§12.4, p 472-474 including Fig. 12.21; then Mitchell (1997), Fig. 8.1, p 233 which shows how the results of nearest neighbor can be different from $k = 5$ nearest ones)

ii. Locally-weighted kernel regression: e.g., estimating house prices as function of age and living area from similar houses (Section 1 of notes based on Mitchell (1997), §8.3.1 p 237-238; `k_NN_kernel_regression_example.m/py`)

iii. (Time permitting:) Using PCA for dimensionality reduction to avoid curse of dimensionality when looking for nearest neighbors in a high-dimensional space. (Section 2 of notes)

iv. k-NN application: the Netflix challenge part II (presentation by Atul S. Kulkarni, remote and local links).

(g) (Time permitting) Decision trees:

i. First, definition of entropy in information theory (from Wikipedia, local).

ii. ID3 algorithm: motivation and outline (Mitchell (1997) §3.1-3.3); entropy measure (§3.4.1.1); information gain (§3.4.1.2); ID3 algorithm (Table 3.1, p 56); example (§3.4.2, p 59-61, here).

iii. (Time permitting:) If the potential labeling variable is a continuous number, need to try all possible values to find the one that leads to the maximum entropy gain, as demonstrated for classifying houses into two neighborhoods based on house price and house area in `decision_tree_ID3_information_gain_continuous_label_example.m/py`.

(h) (Time permitting:) Additional issues:

i. Avoiding over-fitting, pruning and dealing with continuous variables and thresholds (Mitchell (1997), §3.7).

ii. C4.5 algorithm for decision trees (Mitchell (1997), §3.7)

iii. Fisher's Linear discriminant analysis (LDA) leading to a generalized eigenvalue problem (notes)

iv. From binary classification (two classes) to multiple classes: one vs rest and one vs one strategies (here)

v. From linear to nonlinear classification, the kernel trick.

vi. Nearest neighbors using k-d trees.

8. Review. Sources. Notes: chapter 8.

References

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Kavraki, L. E. (2007). Molecular distance measures. OpenStax-CNX module: m11608, Version 1.23: Jun 11, 2007, http://cnx.org/content/m11608/1.23/.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, 42(8):30–37.

Leskovec, J., Rajaraman, A., and Ullman, J. D. (2014). *Mining of massive datasets*. Cambridge University Press.

Mitchell, T. (1997). *Machine learning*. McGraw-Hill Science/ Engineering/ Math.

Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.

Strang, G. (2006). *Linear algebra and its applications. 4th ed*. Cengage Learning.

Strang, G. (2016). *Introduction to linear algebra. 5th ed*. Wellesley-Cambridge Press.

Strang, G. (2019). *Linear Algebra and Learning from Data*. publisher TBD.

Strogatz, S. (1994). *Nonlinear dynamics and chaos*. Westview Press.

Trefethen, L. N. and Bau III, D. (1997). *Numerical linear algebra*, volume 50. Siam.

Von-Luxburg, U. (2007). A tutorial on spectral clustering. *Statistics and computing*, 17(4):395–416.

Vozalis, M. G. and Margaritis, K. G. (2006). Applying SVD on generalized item-based filtering. *IJCSA*, 3(3):27–51.