Final Projects

Instructor: Mauro Maggioni
Web page: https://mauromaggioni.duckdns.org **E-mail**: myfirstname.maggioni@jhu.edu

Each project should be completed by a team (1-4 people). Projects may contain different sections - e.g. theory, algorithms, data exploration. Each team will produce a short report (up to 10 pages) and deliver a 5×(number of team members) minutes presentation, with each team member presenting a 5-minutes part. Every team member should be responsible for a part of the project - this should be clearly indicated at the end of the report - and for the 5 minutes of presentation.

The projects described here are fluid in nature, and I am happy to discuss variations, changes, as well as projects different from the ones described below.

List of Projects

- Random Matrices, Matrix Completion and Multi-Task Learning
- Compressed Sensing, Sparse Estimation, Inverse Problems
- Wavelets and their Applications
- Manifold Learning and Diffusion Maps
- Spectral Clustering
- Semisupervised Learning
- Hyperspectral Images
- Text Documents
- PageRank, and search engines
- Dictionary Learning

Random Matrices, Matrix Completion and Multi-Task Learning

Random matrix theory is a broad field that studies properties of random matrices, and in particular their spectrum (the set of eigenvalues). Problems involving random matrices appear in a variety of applications, from Physics (Wigner was inspired by problems in quantum mechanics when he introduced his model for random matrices and studied their spectrum) to signal processing, to numerical linear algebra.

An introduction to random matrices that fits with what we have learned in the course and provides useful tools is in Chapter 4 of R. Vershynin's notes/book *High-Dimensional Probability - An Introduction with Applications in Data Science*. The first goal is to understand statements and techniques of proof of two main results in that chapter, namely Theorem 4.4.5 (Norm of matrices with sub-gaussian entries) and Theorem 4.6.1 (Two-sided bound on sub-gaussian matrices) [please be careful in identifying these Theorems, as different versions of the book/lecture notes may have slightly different numbering], and report on them.

For example:

- In that chapter he considers two basic applications to community detection in networks and covariance estimation. Pick one of the two applications, and report on it, discussing the main ideas, results, techniques of proof, and possible applications. Conduct numerical experiments that verify the theoretical statements. Discuss if and in which cases the results proposed fall short in applications (e.g. which assumptions may be overly restrictive and why), and whether you think some of these results may be extended to hold in greater generality.

- for covariance matrix estimation, it is interesting to consider the case when the matrix is effectively low-rank, and show that the required number of samples for a certain accuracy in estimation in fact depends on a stable notion of rank, see Section 9.2.3 and Theorem 9.2.4 (Covariance estimation for lower-dimensional distributions). Discuss, and conduct numerical tests to verify the result. You may also the chapter on covariance estimation, PCA and sparse PCA in Rigollet's lecture notes.

- spectral clustering: consider geometric graphs, constructed from points in $\mathbb{R}^D$ by connecting each point to its K nearest neighbors. Find conditions where the results above yield guarantees on the performance of spectral clustering. Discuss/explore how spectral clustering in this setting is highly robust with respect to the shape of the clusters.

- matrix completion: see section 6.5 in R. Vershynin's notes and/or Chapter 4 in Rigollet's lecture notes (in which he makes the connections with Multi-Task Learning).

Algorithmic part
Depending on what you focused on above, you could discuss algorithms for randomized linear algebra (for example randomized SVD, which uses heavily random matrix theory and the JL lemma), spectral clustering (graph construction, eigenvector computation), or matrix completion (for example, singular value thresholding as discussed in Rigollet's notes, but there are in fact many references and possibilities here).

Data exploration part
Explore various models of random matrices (either based on the distribution of the entries, or of proximity graphs constructed from random points) and the corresponding spectra/covariance matrices/spectral clustering properties; if you focused on the randomized linear algebra computational experiments with randomized SVD; if you focused on matrix completion/multi-task learning you can experiment with both toy data sets or real ones (of the type of Netflix challenge).

Compressed Sensing, Sparse Estimation, Inverse Problems

We have discussed sparsity in the context of regression, but not its connections with the field of compressed sensing. Chapter 10 of R. Vershynin's notes/book *High-Dimensional Probability - An Introduction with Applications in Data Science* makes several of these connections. In that chapter you may choose to focus on exact recovery (section 10.5): under suitable assumption, ℓ^1-regularized estimators not only perform well, but in fact reconstruct *exactly* the original signal. This is based on an "escape Theorem" with a simple yet powerful geometric interpretation, where Gaussian width/statistical dimension play a fundamental role. Also consider the aspect of restricted isometries, and the fact that (suitably) random matrices satisfy the restricted isometry condition with high probability, complete the picture.

Inverse problems: consider the paper *Solving Inverse Problems with Piecewise Linear Estimators: From Gaussian Mixture Models to Structured Sparsity*, by Guoshen Yu, Guillermo Sapiro, and Stéphane Mallat, `https://arxiv.org/pdf/1006.3056.pdf` (see also the slides at `http://www.cmap.polytechnique.fr/~yu/research/PLE/talk_PLE_v7.pdf` and a version of the paper with proper figures at `https://archive.org/details/arxiv-1006.3056/page/n29`). It involves constructing probablistic models with a combination of Gaussian mixture models and sparsity to solve inverse problems, especially for images (with applications that include denoising, superresolution, inpainting, etc...).

Applications of original compressed sensing to pediatric MRI: `https://onlinelibrary.wiley.com/doi/full/10.1002/mrm.21391` and `https://pubs.rsna.org/doi/abs/10.1148/radiol.10091218`.

Partially motivated by applications to imaging (and MRI), an extended compressed sensing framework was introduced, which is being implemented in MRI machines: *Breaking the Coherence Barrier: a new theory for compressed*

sensing, B. Adcock, A.C. Hansen, C. Poon, B. Roman https://doi.org/10.1017/fms.2016.32, see also the slides at http://www.ee.ucl.ac.uk/sahd2014/resources/Hansen.pdf.

Large collection of references on compressed sensing, organized by sub-topic, is available at http://dsp.rice.edu/cs/

Algorithmic part
There are a multitude of algorithms for compressed sensing, Lasso, etc...You could discuss what the challenges are, and overview different algorithms, or focus on one and discuss its

Wavelets and their Applications

A standard reference for wavelets if the book *A Wavelet Tour of Signal Processing* by S. Mallat, or *Ten Lectures on Wavelets* by I. Daubechies. I will take the former as a references here. Chapter 1 is a nice general introduction. You could focus on the ideas in Chapter 4 (Time meets Frequency), or Chapter 6 (Wavelet Zoom), or Chapter 7 (Wavelet Bases, focusing on the construction of wavelet bases) or Chapter 9 (An Approximation Tour) or Chapter 10 (Estimators are Approximation). You could read one of these chapters and report on the main ideas and applications.

Algorithmic part
The Fast Wavelet Transform (FWT) is a fast algorithms for computing all wavelet coefficients of a signal. Explain what this means (what would the cost of a naive algorithm for computing all the wavelet coefficients? what is the cost of the FWT?). Implementations are available, for example in the wavelet toolbox in Matlab. Matlab also has a user interface with examples (or you may load your own data, one-/two-/three-dimensional) and functionality for taking wavelet transforms, compressing, denoising, etc....Use the commands wavedemo or wavemenu to access this functionality in Matlab with the Wavelet toolbox installed.

Data Exploration part
You may use either examples contained in the Matlab Wavelet toolbox or create your own or use data of interest to you. This could be music files or images.

Manifold Learning and Diffusion Maps

In manifold learning one assumes that data given in high-dimensional spaces in fact is on (or concentrated around) a low-dimensional regular set, namely a manifold. Slides with an introduction and tons of references are available here: http://mlsp2012.conwiz.dk/fileadmin/lectures/mlsp2012_raich.pdf

Theory part
You could focus on understanding the motivations, objectives of manifold learning, focus on 2-3 techniques (e.g. ISOMAP, Local Linear Embeddings, Laplacian eigenmaps/Diffusion maps) explaining the similarities and differences (both in terms of assumptions, theory and algorithms). Diffusion Maps has been applied in a very large variety of settings beyond manifold learning, since it can be adapted to dynamical systems (from molecular dynamics (e.g. *Determination of reaction coordinates via locally scaled diffusion map*, https://aip.scitation.org/doi/10.1063/1.3569857, to single cell trajectories https://www.youtube.com/watch?v=gVAEk2zBf_E) and to signal processing on graphs (e.g. *The Emerging Field of Signal Processing on Graphs*, https://arxiv.org/pdf/1211.0053.pdf).

Algorithmic part
Several toolboxes implementing manifold learning algorithms are available, in various languages (Python, Matlab, R, ...). Discuss the computational complexity of the different algorithms you are considering,

Data Exploration part
Trying the algorithms on simple data sets (usually available with the toolboxes themselves), or even better on some

real data set of interest to you (or ask me - these techniques have a really wide range of applicability that many, many different data sets may be used).

Spectral Clustering

In this project you will learn about spectral clustering and construct your own spectral clustering algorithm, and use it on data sets.

Theory part
A good start is the page of C. Ding: `http://ranger.uta.edu/~chqding/Spectral/`: it contains tutorial slides, and many relevant references. In particular, among others, I would point out the following ones, that may are rather accessible readings, in rough decreasing order of importance as far as the project is regarded.

[1] J. Shi and J. Malik. Normalized cuts and image segmentation. IEEE. Trans. on Pattern Analysis and Machine Intelligence, 22:888–905, 2000
[2] A.Y. Ng, M.I. Jordan, and Y. Weiss. On spectral clustering: Analysis and an algorithm. Proc. Neural Info. Processing Systems (NIPS 2001), 2001.
[3] F.R. Bach and M.I. Jordan. Learning spectral clustering. Neural Info. Processing Systems 16 (NIPS 2003), 2003.
[4] M. Belkin and P. Niyogi. Laplacian Eigenmaps and Spectral Techniques for Embedding and Clustering, Advances in Neural Information Processing Systems 14 (NIPS 2001), pp: 585-591, MIT Press, Cambridge, 2002.

Write a summary on spectral clustering. There are different flavors, but pick one and describe it in terms of its objective, and how it constructs clusters from initial data. You may use the version we described in class, or any other flavor from the above or any other reference (which you should cite) that you find and like.

Algorithmic part
Write a summary summarizing an algorithm of your choice for spectral clustering. It should state clearly what the inputs, outputs of the algorithm are, and ideally (if you can) how expensive in terms of computations the algorithm is.
Implement the algorithm you discussed above in Matlab (if you prefer a different language, ask me in advance; it should be something I, and anybody else, should be able to run easily). It may be useful to use the Diffusion Geometry toolbox to do so, for example for construing nearest neighbor graphs and computing eigenvectors of the Laplacian or related matrices. Discuss, at least quantitatively or by experimentation, how computationally and memory-intensive the algorithm is. You may neglect in this discussion the time it takes to construct the graph, and especially how long it takes to compute nearest neighbors, as it is hard to estimate that theoretically, albeit you can measure this in practice and comment on it.

Data Exploration part
Test your algorithms on at least 3 synthetic examples, qualitatively different from each other, some low-dimensional, some high-dimensional, where you expect the algorithm to work well, in terms of finding the right clusters. Do at least 1 or 2 synthetic examples, qualitatively different from each other, where you expect the algorithms to not work well. Do the results of running the algorithm match your expectations? Yes/no and why? Measure the performance of the algorithm using a meaningful notion of performance (for how well the algorithm finds the correct clusters). For all of the examples, compare the results with those obtained by running K-means clustering, and comment on the results.
Perform spectral clustering on the following subsets MNIST digits data set (some of these subsets may be too large, depending on your implementation, in which case you may want to extract a subset of the data - indicate clearly when and how you do that), or other data sets linked at on the course wiki. The first subset consists of all 1's and 0's, the second subset of all 1's and 7's, the third subset of all 1's, 3's, 6's, 0's, and the final subset is the set of all digits. For each of them look at the K clusters obtained (with K chosen equal to the number of classes in the subset, i.e. $2, 2, 4, 10$ respectively) and measure in a meaningful way how "pure" each of them is, i.e. how close it is to containing only points with the same label.
Finally, add noise to the data, e.g. Gaussian noise with different standard deviations, and discuss how the results of the above change.

Semisupervised learning

In seminsupervised learning, as discussed in class, one has a large number of unlabeled data points $\{x_i\}_{i=1}^n \subset \mathbb{R}^D$, and a small subset of m of them, which without loss of generality we may assume to be the first m, are labeled, i.e. we are also given $y_1 := f(x_1), \ldots, y_m := f(x_m)$, the labels of $x_1, \ldots, x_m$. Assume that there are L labels $\{1, \ldots, L\}$, so that $y_i \in \{1, \ldots, L\}$ for every i. The idea is that the n points can suggest a good way of representing functions on the data, that we can then use to learn a good classifier $\hat{f}$. In this setting $\hat{f}$ will be a function defined on $x_1, \ldots, x_n$ only (instead of a function defined on the whole space $\mathbb{R}^D$), and $\hat{f}(x_i) \in \{1, \ldots, L\}$ is the label of x_i. We measure the error incurred by a classified $\hat{f}$ on the unlabeled data, so we define the misclassification error rate as

$$err(\hat{f}) := \frac{\#\{\hat{f}(x_i) \neq f(x_i), i = m+1, \ldots, n\}}{n-m}.$$

Theory part
There are several papers that have look at semi supervised learning. Here are some, in rough order of decreasing level of accessibility:

[1] A general framework for adaptive regularization based on diffusion processes on graphs , A.D. Szlam, M. Maggioni, R.R. Coifman. J.M.L.R., 9 (2008) 1711-1739. This is available on my web page.
[2] M. Belkin. Problems of learning on manifolds. PhD thesis, University of Chicago, 2003.
[3] M. Belkin and P. Niyogi. Using manifold structure for partially labelled classification. Advances in NIPS 15, 2003a.
[4] M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In Advances in Neural Information Processing Systems 14 (NIPS 2001), pages 585?591. MIT Press, Cambridge, 2001.
[5] Aarti Singh, Robert D. Nowak, Xiaojin Zhu. Unlabeled data: Now it helps, now it doesn't. NIPS 2008, `http://pages.cs.wisc.edu/~jerryzhu/pub/NIPS08_SSL_v6.pdf`.

While you may choose any of the semi supervised learning constructions discussed in the above references (or elsewhere, but consult me first in this case), but I suggest you use either the one in [3] or the simple version of the ones discussed in [1], where there is no "function adaptation", i.e. the similarity kernel is chosen "in the usual way" as $W(x,y) = e^{-||x-y||^2/\sigma}$ (for example as in equation (3) in the version of [1] on my web page (not the shorter JMLR version), or in equation (14) but with $\sigma_2 = +\infty$ so that there is no "function adaptation"). You may choose and both compare, that would give you extra points.
Write a description of the construction(s), and its (their) motivations, that you decide to use

Algorithmic part
Implement the construction of the classifier. You may use the Diffusion Geometry toolbox for constructing nearest-neighbor graphs, random walks matrices, eigenvectors of the Laplacian etc...The rest of the code has to be yours. Also write code such that given the input as described above, outputs the classifier $\hat{f}$ and the predicted labels $\hat{f}(x_i)$, $i = m+1, \ldots, n$, as well as the error rate defined above.
Discuss, at least quantitatively or by experimentation, how computationally and memory-intensive the algorithm is. You may neglect in this discussion the time it takes to construct the graph, and especially how long it takes to compute nearest neighbors, as it is hard to estimate that theoretically, albeit you can measure this in practice and comment on it.

Data Exploration part
Pick any of the data sets available/linked to from the wiki that are suitable for classification tasks (e.g. MNIST handwritten digits), and test your classifier on at least two of them, and study the performance (in terms of the error rate above) of the classifier as n, m vary. Your algorithm will also have a small number of parameters (e.g. the scale σ in the kernel, and if you are using random walks, there will be parameter t for the length of the random walk): clearly state what these parameters are, how you expect they should be chosen, study how the performance of the algorithm changes as you change them.
(**) You may choose these parameters by hiding part of the known labels, measure performance of $\hat{f}$ on the labels you had but hid, and pick the parameters that maximizing the performance of $\hat{f}$. Study how this works in the examples you picked.

Hyperspectral images

Theory part You will analyze hyperspectral images. A hyperspectral image I is indexed as $I(x, y, k)$, with two spatial indices x, y and a spectral index k. In other words, at every pixel (x, y) a high-dimensional spectral vector $s(x, y) \in \mathbb{R}^S$. The k-th entry of $s(x, y)$ represents the number of photons of light in the frequency bin $f(k)$ at position (x, y). $f(k)$ is a frequency bin that depends on the instrument, and typically ranges from visible to near-infrared. The idea is that the spectrum $s(x, y)$ depends on the chemical composition of what is present at location (x, y) (this is only an approximation, as multiple locations in the scene may contribute to a given pixel (x, y). Applications of hyperspectral imaging are in satellite imaging, agriculture, medical devices.

A simple model for these images is the following: suppose $s_1, \dots, s_k \in \mathbb{R}^S$ are the spectra, called *signatures*, corresponding to k different materials in a scene. Then one assumes that one may write

$$s(x, y) = \alpha_1 s_1 + \cdots + \alpha_k s_k \tag{0.1}$$

where α_i is the amount of material i at position (x, y). There are many reasons why this is a highly simplified model (can you say why?), but we will go with it anyways.
We will consider the following situation: we are given a hyperspectral image I, and we are not given $s_1, \dots, s_k$, and in fact we may not even know k. The problem is then the following: given I, find candidates for $k \in \{1, \dots\}$ and for $s_1, \dots, s_k \in \mathbb{R}^S$, and then deduce the coefficients $\alpha_i(x, y)$, $i = 1, \dots, k$, for every pixel.
We shall consider two methods:

- SVD-based: if the model above is correct, then performing SVD on the collection of spectra $S := \{s(x, y)\}_{x,y} \subset \mathbb{R}^{S \times n}$, where n is the number of pixels, should reveal k, the rank of the set of spectra, as well as yield an orthonormal basis $u_1, \dots, u_k$ for the subspace spanned by the vectors in S. We may let $s_i := u_i$, for $i = 1, \dots, k$, and it is then easy to find the $\alpha_i(x, y)$ corresponding to each pixel. Explain in detail these statements.

- NMF-based: the technique above in general does not yield results satisfying natural physical constraints, in particular since $s(x, y)$ is proportional to a photon count, it should be a nonnegative number. Nonnegative Matrix Factorization (NMF) factorizes the matrix $S \in \mathbb{R}^{S \times n}$ of n (=number of pixels) spectra in $\mathbb{R}^S$ as the product of a matrix of signatures $A \in \mathbb{R}^{S \times k}$ and a matrix of coefficients $W \in \mathbb{R}^{k \times n}$, with both A and W having nonnegative entries. It is not quite clear how to find k, you will need to run the algorithm multiple times with different values of k, and come up with a criterion for deciding which choice of k is "best". Interpret this factorization as giving a decomposition as in (0.1) with nonnegative signatures and coefficients, and explain why this has a physical interpretation.

Algorithmic part
Implement both the SVD-based and the NMF-based approaches (you may use existing toolboxes/codes for calculating NMF). If you are using Matlab, note there are functions for both types of factorizations. The SVD may scale to pretty large matrices if computed correctly (explore the use of the "economy" form of the SVD), while NMF is much more expensive and it will not scale to large data sets. In order to scale the algorithms to larger data sets, consider the use of random samples: instead of running the algorithm on all the spectra in an image, select a random subset of n_0 samples (with $n_0 \ll n$, where n is the number of pixels) and run the algorithms on those n_0 spectra to obtain k and the signatures $s_1, \dots, s_k$. Experiment to see how the results (both k and $s_1, \dots, s_k$) change for different values of n_0. Does it seem necessary to pick n_0 quite large (e.g. a fraction of n) or n_0 quite small seems to suffice?
Add noise to the images, e.g. Gaussian noise with different standard deviations, and discuss how the results of the above change.

Data Exploration part
I have uploaded a few hyperspectral images on the wiki: run the above experiments on any or all the images there. Now consider the following problems, related to the measurements being used. Let's call $e_1, \dots, e_S$ the standard basis for the spectral space $\mathbb{R}^S$. The instrument as described above measures for each pixel, the S frequency bins, which we could model as $(\langle s(x, y), e_i \rangle)_{i=1,\dots,s}$.

- Suppose the instrument can be designed so that instead of measuring these coefficients, can measure $(\langle s(x, y), v_i \rangle)_{i=1,\dots,s'}$, where the v_i's may be specified once for all, and having $s' \ll s$ is beneficial (e.g. less costly instrument, or less

noisy measurements). Make a proposal on how to choose the v_i's. This may use, or not, the signatures found by any of the algorithms above

- Consider the case where the v_i's are random Gaussian vectors, and/or random ± 1 vectors, and/or random $\{0,1\}$ vectors. How does the performance of the algorithms above seem to change as s' varies from 1 to s?

Text documents

Theory part

We consider a corpus of documents in the Science News data set (uploaded to the wiki page for the course), and the Newsgroup data set `http://qwone.com/%7Ejason/20Newsgroups/`. Each document may be mapped (but you are welcome to consider other ways you may find in the literature) to a high dimensional vector as follows - this is called the bag-of-words model. Collect all the words in the documents in a dictionary $\mathcal{D}$ of size D', then map each document to a vector in $\mathbb{R}^{D'}$ where the k-th coordinate is the number of times that the k-th word in the dictionary appears in that document. If there are n documents, this will yield a data set of n points in D' dimensions, that may collected in a $D' \times n$ word-document matrix. Before proceeding, you may want to clean up and normalize the data. You may want to discard all words that appear too often and/or too uniformly across documents (e.g. words such as "a", "the", "that", etc...may be among these), as well as words that appear too rarely. In order to measure uniformity, consider a measure of entropy: if you look at row k of the data matrix (let us denote it by r_k) you have constructed, the j-th entry represents how many times the k-th word of the dictionary has appeared in document j. You may transform r_k to a probability vector p_k with $\sum_j p_k(j) = 1$ by multiplying by an appropriate constant. Then consider the notion of entropy of a probability vector/distribution, defined by $\mathrm{ent}(p_k) = -\sum_j p_k(j) \log p_k(j)$ (with $0 \times \log 0$ is defined to be 0): look this up and study its properties, and describe why large entropy corresponds to a notion of "almost uniform" distribution (and conversely, small entropy corresponds to....?). Use the notion of entropy to decide which words are "too uniformly distributed" to be informative, and therefore should be discarded. At this point the dimension of the data has decreased to $D \leq D'$. You may also want to normalize by the length of the document (albeit most articles are very roughly the same length): for example, before or after (which works best?) discarding words as above, you may normalize the vector x_i associated to the i-th document by multiplying it by a scalar α_i so that $||\alpha_i x_i||_1 := |\alpha_i| \sum_{k=1}^{D} |(x_i)_k| = 1$, which has the interpretation that the new k-th coordinate is the *frequency* of the k-th word in the dictionary in document i. Another possibility is to normalize x_i so that $||\alpha_i x_i||_2^2 := \sum_{i=1}^{D} |\alpha_i x_i|^2 = 1$.

Another aspect of the "theory part" is reading about Latent Semantic Indexing, and understanding its connections with PCA, and its uses for organizing, obtaining low dimensional embeddings, and querying a large set of documents. An introduction, with several core references, is `http://www.cse.msu.edu/~cse960/Papers/LSI/LSI.pdf`.

Algorithmic part

Construct a nearest neighbor graph using the Diffusion Geometry package where we connect each document to its K nearest neighbors. You will explore the dependency of what follows on the choice of K, and also explore the "autotune" option in the Diffusion Geometry package for constructing these graphs.

Embed the data in d dimensions using the eigenvectors of the random walk (or the Laplacian), and run K-means clustering to identify K clusters. Match the K clusters to the L labels, where L is the number of different labels of the documents, and vary K between L and, say, $2L$ (why?). Construct a confusion matrix $L \times L$, where the (l_1, l_2) entry is the frequency of classify a document with true label l_1 as having label l_2 by the label assignment from the clustering algorithm. Compare the results with those of K-means in the original D-dimensional space, and under different normalizations and different pre-processing of the data. For each of the clusters, find the words that are most representative (e.g. most frequent) in each cluster.

Consider now a (supervised) classifier: Support Vector Machines - good references are:

[1] *Support Vector Machines Tutorial*, J. Weston, `http://www.cs.columbia.edu/~kathy/cs4701/documents/jason_svm_tutorial.pdf`.

[2] *A Tutorial on Support Vector Machines for Pattern Recognition*, C. J.C. Burges, `http://research.microsoft.com/pubs/67119/svmtutorial.pdf` [more mathematical].

Greatly summarize how SVM's work, and apply SVM's to the data above, using linear (or at most quadratic) kernsl, and both in original high-dimensional space or in d-dimensional diffusion space (i.e. after embedding with

eigenfunctions of the Laplacian. Note that, as usual in learning, this time you need to create a training set (of size, say, 25%,50% and 75% of the data) on which you construct the classifier, and a test set (the remaining portion of the data) on which you test the classifier, and measure the error (using the confusion matrix, and the total misclassification error). This needs to repeated for multiple (random) choices of splits into training and test data.

Data Exploration part
As mentioned above, explore the Science News data set (on the wiki) and ideally also the Newsgroup Data set (linked at on the wiki), or a subset thereof if the algorithms have too much trouble running on the whole data set (you may pick one or more subsets, each consisting of just 2 or 3 categories in the Newsgroup Data set).

Pagerank and search engines

Theory part
The page `http://infolab.stanford.edu/~backrub/google.html` contains both part of the history and motivation for the construction of the first search engines, Google in particular. The key paper is: [Page 98] Lawrence Page, Sergey Brin, Rajeev Motwani, Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Available at `http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf`. The wikipedia page `https://en.wikipedia.org/wiki/PageRank` also seems quite extensive and clear, with several very relevant references.

Understand the PageRank algorithm, write a summary of its motivation, basic implementation, computational cost, and example usage, and relate it to what we discussed in class about random walks, and their stationary distribution and top eigenvectors. Note and think about how much difference it makes the fact that the citation/link structure of web pages changes the behavior of the random walk, and why PageRank required the extra "teleportation" step.

Algorithmic part
Implement a simple version of the PageRank algorithm. Make sure your implementation uses sparse matrices, and the eigenvector computation is done iteratively, requiring only matrix-vector multiplications, so that large network where every node is connected to a small number of nodes can be analyzed with your code.

Data Exploration part
Pick a network data set from `http://snap.stanford.edu/data/` and apply PageRank. Citation networks, or wikipedia networks (and subsets thereof) are example good data sets to be analyzed. Some of the data sets have networks that change in time (e.g. by considering citations before a certain year t, and then changing t), and one may ask about how the top ranked node change with time, or how the top p nodes change in time.

Dictionary Learning

Theory part
The problem of dictionary learning may be roughly be summarized as follows: given signals $\{x_i\}_{i=1}^n$ in $\mathbb{R}^D$ (for example a short window containing D samples of digital sound, or a little square patch containing D pixels in an image), one seeks to construct a dictionary $\{\varphi_j\}_{j=1}^m$ of vectors in $\mathbb{R}^D$ such that for every i (or, in fact, hopefully for every future signal in same "class" as the x_i's) there exist coefficients $\{\alpha_{ij}\}_{j=1}^m$ such that $x_i \approxeq \sum_{j=1}^m \alpha_{ij}\varphi_j$, and many α_{ij}'s (say all except k of them) are equal to 0, or very small, i.e. $||(\alpha_{ij})_{j=1}^m||_0 := \#\{ j : \alpha_{ij} \neq 0\}$ is small (e.g. equal to $k \ll m$) for every i.

The main motivation comes from statistical signal processing: if one has a dictionary that sparsifies the data, in the sense that every data point has a representation in terms of the dictionary that only requires a small number of nonzero coefficients, then one can compress the data efficiently (by transmitting only the small number of nonzero coefficients for every signal), and can also perform a wide variety of useful statistical tasks efficiently, for example denoising a noisy signal, or "filling in" missing parts of a signal.

Dictionary learning has been an active area of research for 10+ years now. You will explore the area by looking at one of the most popular algorithms, called K-SVD. The main reference is `http://sites.fas.harvard.edu/~cs278/`

papers/ksvd.pdf

If you are familiar with neural networks, in particular autoencoders, you could use those as well.

Algorithmic part

Several version of code for the K-SVD are available, reputable versions are at http://www.cs.technion.ac.il/~ronrubin/software.html and http://www.cs.technion.ac.il/~elad/software/. Install one of the available packages, make sure it runs properly by running it on one of the available examples, and summarize what the algorithm does, discuss the computational complexity, and the results on a simple example.

If you chose autoencoders, existing neural network packages implement them (and replace K-SVD below by autoencoders).

Data Exploration part

- Construct a simple example, in 1-D, with a class of signals generated by taking short random linear combinations of dictionaries (e.g. pick overcomplete sinusoids/Fourier basis/wavelet bases as your dictionaries, similarly to what done in the main reference above), and test under which conditions (number of samples n given, size m of the dictionary, sparsity level k, parameters of the K-SVD algorithm...) K-SVD recovers the underlying dictionary. Repeat the same experiment when noise is added to to the samples given as input to K-SVD.
- Consider a couple of existing data sets, e.g. MNIST handwritten digits http://yann.lecun.com/exdb/mnist/, or images in one or a few classes of the Caltech 256 database http://www.vision.caltech.edu/Image_Datasets/Caltech256/, and use K-SVD for tasks such as denoising and reconstructing when only noisy partial observations are performed, as in the paper references to above.
- Using K-SVD for classification: pick a data set with at least two categories of object (e.g. the ones above) and pick two categories A and B. Train K-SVD on images in class A and on images in class B separately, obtaining two dictionaries. Then for each new image from A or B, expand it sparsely on either dictionary, and design a classifier that tries to tell the class of the image. This may or not work.