LECTURE NOTES: ADVANCED LINEAR ALGEBRA FOR DATA SCIENCE

KEATON HAMM

Last Update: August 23, 2023

These are Lecture Notes to the MATH 496T / MATH 577 course entitled **Advanced Linear Algebra for Data Science** taught at the University of Arizona in the Spring 2020 semester. The course is designed to be an overview of advanced theoretical and computational Linear Algebra which covers many of the topics that are useful to Data Science and Applied Mathematics but are typically not covered in a first or even second course in Linear Algebra at the undergraduate level. The course also has a heavy applications component where many current Data Science and Machine Learning methods are discussed including graph clustering algorithms, compressed sensing, and dimensionality reduction. The main textbook used for the course is Linear Algebra and Learning From Data by Gilbert Strang [15], though we will point out other scholarly articles and texts used throughout the later portion of the notes.

These notes were done as a collaboration between the students of the course and are edited and curated by the instructor (Keaton Hamm). If you find these notes and they are useful to you, then I will be quite pleased; additionally, if you find any errors, typos, etc. feel free to let me know and I will change them in due time (hamm@math.arizona.edu).

0.1. **Notation.** We will use $\mathbb{R}$ and $\mathbb{C}$ to denote the Real and Complex fields, respectively. For ease of exposition, these notes will primarily be concerned with real-valued matrices, but the essential results hold for either case without change. We will use $\mathbb{R}^n$ to be the n–dimensional vector space over the reals, and will call elements thereof vectors or points interchangeably. We will also consider these elements to be column vectors, so that

$$\mathbb{R}^n := \left\{ \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} : x_i \in \mathbb{R} \right\}.$$

The set of $m \times n$ matrices with real entries will be denoted as

$$\mathbb{R}^{m\times n} := \left\{ \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} : a_{ij} \in \mathbb{R} \right\}.$$

Note that other common notation is $\mathscr{M}_{m,n}(\mathbb{R})$.

Given a matrix $A \in \mathbb{R}^{m\times n}$, we will use the Matlab-friendly notation $A_{i:}$ to denote its i^{th} row and $A_{:j}$ to denote its j^{th} column, so

$$A_{i:} = \begin{bmatrix} a_{i1} & \dots & a_{in} \end{bmatrix}, \qquad A_{:j} = \begin{bmatrix} a_{1j} \\ \vdots \\ a_{mj} \end{bmatrix}.$$

1. Lecture 1: Linear Algebra Review

Goal: The purpose of this lecture is to give a fly-by review of the basic facts about Linear Algebra that are needed for this course. We will not review absolutely everything at the front end so that we can get to some fun stuff sooner, but this lecture recalls a lot of the very basic facts that will be needed in the first few lectures.

1.1. **Basics of Vectors.** $\mathbb{R}^n$ is a **vector space**, which is a collection of vectors along with a notion of *vector addition* and *scalar multiplication*. Addition of two vectors is done coordinatewise:

$$\begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} x_1 + y_1 \\ \vdots \\ x_n + y_n \end{bmatrix},$$

and scalar multiplication is applied to all coordinates, i.e., if $\alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$, then

$$\alpha x = \begin{bmatrix} \alpha x_1 \\ \vdots \\ \alpha x_n \end{bmatrix}.$$

For the full definition of an abstract vector space, see Appendix A.

Definition 1.1. *Given a collection of vectors $\{x_1, \dots, x_k\} \subset \mathbb{R}^n$, a **linear combination** of these vectors is any vector of the form*

$$\alpha_1 x_1 + \dots \alpha_k x_k = \sum_{i=1}^{k} \alpha_i x_i, \quad \alpha_1, \dots, \alpha_k \in \mathbb{R}.$$

Note any such linear combination must be in $\mathbb{R}^n$ by the vector space properties.

Definition 1.2. *Given a collection of vectors $\{x_i : i = 1, \dots, k\}$, its **linear span** is the set of all possible linear combinations:*

$$\operatorname{span}\{x_i : i = 1, \dots, k\} := \left\{ \sum_{i=1}^{k} \alpha_i x_i : \alpha_i \in \mathbb{R} \right\}.$$

Definition 1.3. *A collection of vectors $\{x_1, \dots, x_k\} \subset \mathbb{R}^n$ is a **spanning set** for $\mathbb{R}^n$ provided every $x \in \mathbb{R}^n$ can be written as a linear combination of $x_1, \dots, x_k$, i.e., for every $x \in \mathbb{R}^n$, there exists $\alpha_1, \dots, \alpha_k$ such that*

$$x = \sum_{i=1}^{k} \alpha_i x_i.$$

Note this is equivalent to

$$\operatorname{span}\{x_i : i = 1, \dots, k\} = \mathbb{R}^n.$$

Definition 1.4. *A collection of vectors $\{x_1, \dots, x_k\} \subset \mathbb{R}^n$ is **linearly independent** provided*

$$\sum_{i=1}^{k} \alpha_i x_i = 0 \quad \text{if and only if} \quad \alpha_1 = \dots = \alpha_k = 0.$$

*A collection of vectors is **linearly dependent** if it is not linearly independent (thus there exists $\alpha_1, \dots, \alpha_k$, at least one of which is not zero such that $\sum_{i=1}^{k} \alpha_i x_i = 0$).*

Definition 1.5. *A collection of vectors $\{x_1, \dots, x_k\} \subset \mathbb{R}^n$ is a **basis** for $\mathbb{R}^n$ provided it is linearly independent and a spanning set.*

Theorem 1.6. *Suppose that $X = \{x_1, \dots, x_k\} \subset \mathbb{R}^n$.*

(1) *If X is a spanning set, then $k \geq n$.*
(2) *If X is linearly independent, then $k \leq n$*
(3) *If X is a basis, then $k = n$.*
(4) *X is a basis for $\mathbb{R}^n$ if and only if for every $x \in \mathbb{R}^n$, there exists a unique set of scalars $\alpha_1, \ldots, \alpha_n$ such that $x = \sum_{i=1}^n \alpha_i x_i$.*

Note that Theorem 1.6(3) tells us how to define the dimension of a space.

Definition 1.7. *The **dimension** of a vector space is the size of a basis for it. The dimension of a vector space V is denoted* $\dim(V)$.

Theorem 1.6 and the above definition tell us the important fact that

$$\dim(\mathbb{R}^n) = n.$$

The space of n–dimensional real vectors, $\mathbb{R}^n$, has a **dot product** (also termed an **inner product**) which we will write in a variety of ways:

$$x \cdot y = \langle x, y \rangle = \sum_{i=1}^n x_i y_i = y^T x.$$

For the general definition of an inner product on a vector space, see Appendix A.

Definition 1.8. *Two vectors $x, y \in \mathbb{R}^n$ are called **orthogonal** if $\langle x, y \rangle = 0$.*
*A collection of vectors $\{x_1, \ldots, x_k\} \subset \mathbb{R}^n$ is called **orthogonal** provided $\langle x_i, x_j \rangle = 0$ for all $i \neq j$.*
*A collection of orthogonal vectors is called **orthonormal** provided they satisfy the additional condition that $|x| = 1$, where $|x| := \sqrt{x_1^2 + \cdots + x_n^2}$ is its Euclidean length.*

These definitions bring us to some important facts about collections of orthogonal vectors.

Theorem 1.9. *Suppose that $X = \{x_1, \ldots, x_k\} \subset \mathbb{R}^n$ is orthogonal. Then the following statements hold:*

(1) *X is linearly independent*
(2) *$k \leq n$*
(3) *If $k = n$, then X is a basis for $\mathbb{R}^n$.*

The proof of Theorem 1.9 will be left as a homework exercise for the first lecture.

Note that the inner product contains additional geometric information; namely, the inner product between two unit vectors indicates how close they are to being parallel or orthogonal. If two unit vectors are parallel, then their inner product is 1, if they are anti-parallel (in the opposite direction) then their inner product is -1, and if they are orthogonal then their inner product is 0. Thus the closer $\langle x, y \rangle$ is to ± 1, the closer x and y are to being parallel, and the closer to 0 the closer they are to being orthogonal.

1.2. **Subspaces.** One of the most important notions we will discuss is that of subspaces of a vector space.

Definition 1.10. *A subset $\mathscr{S} \subset \mathbb{R}^n$ is a **subspace** of $\mathbb{R}^n$ if it is a vector space over $\mathbb{R}$ (see Appendix A for the full definition of a vector space).*

A glance at Appendix A shows that one would need to verify 8 conditions of $\mathscr{S}$ to check that it is a subspace of $\mathbb{R}^n$. However, we may reduce the amount of conditions to 3 by the following theorem.

Theorem 1.11. *A subset $\mathscr{S} \subset \mathbb{R}^n$ is a subspace of $\mathbb{R}^n$ if and only if the following conditions hold:*

(1) $\mathscr{S} \neq \emptyset$
(2) $x + y \in \mathscr{S}$ *for all* $x, y \in \mathscr{S}$
(3) $\alpha x \in \mathscr{S}$ *for all* $\alpha \in \mathbb{R}$ *and* $x \in \mathscr{S}$.

That is, $\mathscr{S}$ is nonempty, closed under vector addition and closed under scalar multiplication.

How should we typically think of subspaces? Well, the following is a very important step:

Proposition 1.12. *Let* $\{x_1, \ldots, x_k\} \subset \mathbb{R}^n$. *Then* $\operatorname{span}(x_1, \ldots, x_k)$ *is a subspace of* $\mathbb{R}^n$.

The proof is left as an exercise that should be done by the judicious reader.

Now, since subspaces are vector spaces themselves, they must have a dimension. So what can we say about the dimension of the span of a collection of vectors?

Theorem 1.13. *If* $X = \{x_1, \ldots, x_k\} \subset \mathbb{R}^n$ *is linearly independent, then* $\dim(\operatorname{span}(X)) = k$.

It is important to note that a necessary condition for X to be linearly independent in the preceding theorem is that $k \leq n$, and it is an important fact to note that if $\mathscr{S} \subset \mathscr{V} \subset \mathbb{R}^n$ and $\mathscr{S}$ is a subspace of $\mathscr{V}$ which is a subspace of $\mathbb{R}^n$, then

$$\dim(\mathscr{S}) \leq \dim(\mathscr{V}) \leq \dim(\mathbb{R}^n) = n.$$

1.3. **Basics of Matrices.** We may denote real-valued matrices $A \in \mathbb{R}^{m \times n}$ in a variety of ways, for example via its scalar entries $A = [a_{i,j}]_{i=1}^{m}{}_{j=1}^{n}$ or by its columns $A = [A_{:1} \ldots A_{:n}]$.

Note that many data sets can be naturally represented as a matrix, and thus are typically thought of as an array of numbers, but we will see that it is of great utility to consider the linear algebraic structure of such data matrices rather than simply considering them as a collection of scalars.

First, let's discuss matrix–vector multiplication. Given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $x \in \mathbb{R}^n$, the product Ax is a vector in $\mathbb{R}^m$ whose i–th coordinate is given by

$$(Ax)_i = \sum_{j=1}^{n} a_{ij} x_j = \langle A_{i:}, x \rangle .$$

That is, the i–th entry of Ax is the inner product of the i–th row of A with x.

However, we can also write Ax as a linear combination of its columns!!

$$Ax = \sum_{j=1}^{n} x_j A_{:j}.$$

Recall here that $A_{:j} \in \mathbb{R}^m$ is the j–th column of A.

So we can either evaluate Ax via inner products or linear combinations of columns of A. One should note that both methods of computation require mn multiplications, so neither is necessarily better from that viewpoint.

Now for matrix–matrix multiplication. We may do this in two ways that both have useful information embedded in them. First of all, we recall that for the matrix–matrix product AB to be well defined, B must have exactly as many rows as A has columns. So if $A \in \mathbb{R}^{m \times n}$ and $B \in \mathbb{R}^{n \times p}$, then $AB \in \mathbb{R}^{m \times p}$. If B has more or less than n rows, then the product is not defined.

Matrix Multiplication via Inner Products: The first way we can write the product is as inner products similar to what we did for matrix–vector products:

$$(AB)_{ij} = \langle A_{i:}, B_{:j} \rangle = \sum_{k=1}^{p} A_{ik} B_{kj}.$$

Matrix Multiplication via Outer Products: Alternatively, we may express the matrix–matrix product by what are called *outer products* of two vectors, namely

$$AB = \sum_{k=1}^{p} A_{:k} B_{k:}. \tag{1}$$

Note here that $A_{:k}$ is a column vector with dimension $(m \times 1)$ and $B_{k:}$ is a row vector with dimension $(1 \times p)$, hence their product has dimension $(m \times p)$ as we expected. To see that this is equivalent to the first expression, note that if we want the (i, j)–th entry of AB from the outer product form we need to evaluate $\sum_{k=1}^{p} A_{ik} B_{kj}$ which is $(AB)_{ij}$.

Equation (1) gives us an interesting first glimpse at the fundamental notion of a matrix being represented as the sum of rank one matrices, which are the building blocks of matrix spaces in general. To wit, let us now discuss the notion of the rank of a matrix.

Definition 1.14. *Let $A \in \mathbb{R}^{m \times n}$.*

*(1) The **column space** of A is $\mathrm{Col}(A) := \mathrm{span}(A_{:1}, \ldots, A_{:n})$ which is a subspace of $\mathbb{R}^m$.*
*(2) The **row space** of A is $\mathrm{Row}(A) := \mathrm{span}(A_{1:}, \ldots, A_{m:})$ which is a subspace of $\mathbb{R}^n$.*

A fundamental fact of Linear Algebra is the following.

Theorem 1.15. *For every $A \in \mathbb{R}^{m \times n}$,*

$$\dim(\mathrm{Col}(A)) = \dim(\mathrm{Row}(A)).$$

This leads to a well-defined notion of the rank of a matrix.

Definition 1.16. *Given $A \in \mathbb{R}^{m \times n}$, the **rank** of A is given by*

$$\mathrm{rank}(A) := \dim(\mathrm{Col}(A))$$

or equivalently (on account of Theorem 1.15) by $\mathrm{rank}(A) = \dim(\mathrm{Row}(A))$.

One observation is immediate since the dimensions of the row and column space cannot be more than m or n, respectively:

$$\mathrm{rank}(A) \leq \min\{m, n\}.$$

We also have the useful facts that

$$\mathrm{Col}(A) = \mathrm{Row}(A^T), \qquad \text{and} \qquad \mathrm{Row}(A) = \mathrm{Col}(A^T).$$

2. Lecture 2: The Four Fundamental Subspaces

Authors' Note: The title of this section is shamelessly copied from Gilbert Strang's text [15] used in the course; we don't try to fix what isn't broken.

Goal: The purpose of this lecture is to illustrate the fundamental subspaces related to a matrix viewed as a linear operator, and to parlay this into a discussion of matrix factorizations.

2.1. The Four Subspaces.

Here, we will discuss the fundamental subspaces that are determined by a matrix $A \in \mathbb{R}^{m\times n}$. Note that such a matrix defines a linear operator from $\mathbb{R}^n \to \mathbb{R}^m$. The first subspace we will consider is the nullspace of A.

Definition 2.1. *Given $A \in \mathbb{R}^{m\times n}$, its **nullspace** is*

$$\mathcal{N}(A) := \{x \in \mathbb{R}^n : Ax = 0\}.$$

Some basic properties of nullspaces are as follows:

Proposition 2.2. *Suppose $A \in \mathbb{R}^{m\times n}$ has rank r. Then the following hold:*

(1) $\mathcal{N}(A)$ is a subspace of $\mathbb{R}^n$
(2) $\dim(\mathcal{N}(A)) = n - r$
(3) $\mathcal{N}(A^T A) = \mathcal{N}(A)$
(4) $\mathcal{N}(A)$ is orthogonal to $\mathrm{Row}(A)$.

Proof. (1): By Theorem 1.11, we need only show that $\mathcal{N}(A)$ is nonempty, and closed under addition and scalar multiplication. First, it is nonempty because $0 \in \mathcal{N}(A)$ (of course $A0 = 0$. Now suppose $x, y \in \mathcal{N}(A)$, then by distributivity of matrix–vector multiplication, we have

$$A(x + y) = Ax + Ay = 0 + 0 = 0,$$

which implies $x + y \in \mathcal{N}(A)$. Now let $x \in \mathcal{N}(A)$ and $\alpha \in \mathbb{R}$, then

$$A(\alpha x) = \alpha Ax = \alpha 0 = 0,$$

whence $\alpha x \in \mathcal{N}(A)$. Putting these observations together, we have that $\mathcal{N}(A)$ is a subspace of $\mathbb{R}^n$.

(2): The proof of this is a bit more involved, and will be relegated to the Appendix. We will also give a more general version of this shortly.

The proof of (3) and (4) are left as a homework exercise. □

Example 2.3. Let

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & -1 \end{bmatrix}.$$

Let's compute $\mathcal{N}(A)$.

We need to find all $x \in \mathbb{R}^3$ such that $Ax = 0$. So letting $x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix}^T$, we have

$$Ax = \begin{bmatrix} 1 & 2 & 3 \\ 1 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} x_1 + 2x_2 + 3x_3 \\ x_1 + 2x_2 - x_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Note that this gives us a set of two equations to solve, and we cah subtract the second from the first to get

$$4x_3 = 0,$$

implying that we must have $x_3 = 0$. Now, it follows that $x_1 = -2x_2$. So any vector of the form

$$\begin{bmatrix} -2\alpha \\ \alpha \\ 0 \end{bmatrix}$$

is in $\mathcal{N}(A)$. For simplicity, we write

$$\mathcal{N}(A) = \text{span}\left(\begin{bmatrix} -2 \\ 1 \\ 0 \end{bmatrix}\right).$$

It will be a homework exercise to repeat a similar example to this and find $\mathcal{N}(A^T)$, which is sometimes called the **left nullspace** of A.

Now how are all of these subspaces related exactly? It terms out that they naturally decompose the spaces $\mathbb{R}^m$ and $\mathbb{R}^n$. To see why, we need a bit more background first.

2.2. **Aside: Orthogonal Subspaces.**

Definition 2.4. *Let $\mathcal{S}$ be a subspace of $\mathbb{R}^n$. The **orthogonal complement** of $\mathcal{S}$ is*

$$\mathcal{S}^\perp := \{x \in \mathbb{R}^n : \langle x, y\rangle = 0, \text{ for all } y \in \mathcal{S}\}.$$

Here are some important properties of orthogonal complements.

Theorem 2.5. *Let $\mathcal{S}$ be a subspace of $\mathbb{R}^n$. Then the following properties hold:*

(1) $0 \in \mathcal{S}^\perp$

(2) $\mathcal{S}^\perp$ *is a subspace of* $\mathbb{R}^n$

(3) $\left(\mathcal{S}^\perp\right)^\perp = \mathcal{S}$

(4) Any $x \in \mathbb{R}^n$ can be written uniquely as $x = y + z$ for some $y \in \mathcal{S}$ and $z \in \mathcal{S}^\perp$

Moreover, if $A \in \mathbb{R}^{m\times n}$, then

(1) $\mathcal{N}(A)^\perp = \text{Row}(A)$

(2) $\mathcal{N}(A^T)^\perp = \text{Col}(A)$

Note that if V and W are two subspaces for which property (4) holds, then we write that $V \bigoplus W = \mathbb{R}^n$. Note also that Theorem 2.5(4) is sometimes called the *Orthogonal Decomposition Theorem*.

Additionally, our considerations here give the essential ingredients to the *Fundamental Theorem of Linear Algebra*, which is also called the *Rank–Nullity Theorem*:

Theorem 2.6 (Fundamental Theorem of Linear Algebra)**.** *Let $A \in \mathbb{R}^{m\times n}$. Then*

$$\dim(\mathcal{N}(A)) + \text{rank}(A) = n$$

and

$$\dim(\mathcal{N}(A^T)) + \text{rank}(A) = m.$$

2.3. **Back to the Four Subspaces.** The "moreover" part of Theorem 2.5 now allows us to draw the most important picture we will have to keep in mind: Figure 1.

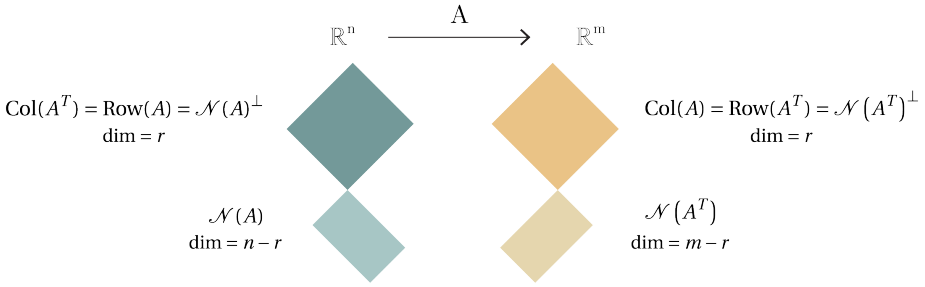

FIGURE 1. Illustration of the Four Fundamental Subspaces of a given rank r matrix $A \in \mathbb{R}^{m\times n}$. Idea copied from Gilbert Strang's textbook [15]; figure produced by Amy Hamm Design.

A matrix $A \in \mathbb{R}^{m\times n}$ maps $\mathbb{R}^n \to \mathbb{R}^m$ as we discussed before. The Orthogonal Decomposition Theorem along with other parts of Theorem 2.5 shows that we may write $\mathbb{R}^n = \mathcal{N}(A) \bigoplus \text{Row}(A)$ and $\mathbb{R}^m = \mathcal{N}(A^T) \bigoplus \text{Col}(A)$ as shown in the figure.

Figure 1 demonstrates the Rank-Nullity Theorem; indeed, on the left-hand side we see that $\dim(\mathcal{N}(A)^\perp) + \dim(\mathcal{N}(A)) = \text{rank}(A)$, while on the right-hand side, we have that $\dim(\mathcal{N}(A^T)^\perp) + \dim(\mathcal{N}(A^T)) = \text{rank}(A)$.

We list several other important properties of ranks in the following theorem.

Theorem 2.7. *Assume that A and B are matrices such that the following operations make sense (their size may vary from line to line). Then the following properties hold:*

(1) $\text{rank}(AB) \le \min\{\text{rank}(A), \text{rank}(B)\}$
(2) $\text{rank}(A+B) \le \text{rank}(A) + \text{rank}(B)$
(3) $\text{rank}(A^TA) = \text{rank}(AA^T) = \text{rank}(A)\text{rank}(A^T)$
(4) If $A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}$ *and* $\text{rank}(A) = \text{rank}(B) = r$, *then* $\text{rank}(AB) = r$
(5) ***Sylvester's Inequality:*** *If* $A \in \mathbb{R}^{m\times r}, B \in \mathbb{R}^{r\times n}$, *then*

$$\text{rank}(A) + \text{rank}(B) \le \text{rank}(AB) + r.$$

Note that Sylvester's Inequality places no restriction on rank(A) or rank(B); these values do not have to be equal, and the matrices do not have to be full rank.

The proof that (5) implies (4) in Theorem 2.7 is left as a homework exercise.

3. Lecture 3: Matrix Decompositions

3.1. **Matrix Factorizations.** In general, we say that a matrix A $\in \mathbb{R}^{m\times n}$ is factored (alternatively, decomposed) via two matrices $B \in \mathbb{R}^{m\times r}$ and $C \in \mathbb{R}^{r\times n}$ if $A = BC$. There are many useful ways to factorize a matrix, each of which have benefits and drawbacks either theoretically or computationally. Many of the methods detailed below may be familiar to the reader from previous courses in Linear Algebra. This section gives a relatively broad introduction to matrix factorizations, whereas subsequent lectures will focus on more pertinent examples to Data Science such as the Singular Value Decomposition.

3.1.1. *Diagonalization.* Diagonalization decomposes a matrix A $\in \mathbb{R}^{n\times n}$ into the product of matrices $S\Lambda S^{-1}$, where S,$\Lambda \in \mathbb{R}^{n\times n}$. In this expression, S is the matrix of the eigenvectors of A and Λ is the diagonal matrix of its eigenvalues (more on these ideas in Section 4).

The value of this decomposition is twofold. Once this diagonalization is known, the eigenvectors and eigenvalues can easily be read off as the columns of S and the diagonal values of Λ. Secondly, diagonalization can be used to avoid taking the products A^TA or A^n directly, as these are expensive computations. To see how, note that A^2 can be written using the diagonalization $A = S\Lambda S^{-1}$ as

$$A^2 = AA = S\Lambda S^{-1}S\Lambda S^{-1}S\Lambda S^{-1} = S\Lambda^2 S^{-1}.$$

This generalizes to

$$A^n = S\Lambda^n S^{-1}.$$

This is almost always a less computationally expensive way to compute A^n than the direct approach.

3.1.2. *The LU Decomposition.* LU stands for Lower Upper, and this type of decomposition factors $A \in \mathbb{R}^{n\times n}$ into the product of $n \times n$ matrices L and U which are **lower triangular** (having all zeros above the diagonal) and **upper triangular** (having all zeros below the diagonal), respectively.

When is this useful? Consider the common problem of solving a system of linear equations, represented in matrix–vector multiplication form as

$$Ax = b$$

where $A \in \mathbb{R}^{n\times n}$ and $b \in \mathbb{R}^n$ are known, and $x \in \mathbb{R}^n$ is unknown. A standard approach to solve this system is to use Gaussian elimination. However, Gaussian elimination can be computationally expensive, requiring many elementary row operations. Instead, this problem can be solved via LU decompositions by considering the following coupled system of equations:

$$Ly = b$$
$$Ux = y$$

That is, we first solve $Ly = b$ for y, and then solve $Ux = y$ for x. These calculations may be done quickly due to the triangular forms of L and U. In fact, each value y_i in the vector y can be calculated by forward and backward substitution:

$$y_i = \frac{b_i - \sum_{k=1}^{i-1} \ell_{i,k} y_k}{\ell_{i,i}}.$$

The entries of x may be computed similarly.

Many other useful decompositions exist, including QR decompositions, CUR decompositions, and Singular Value Decompositions. We will talk about these in later sections.

3.2. **Orthogonal Matrices.** Any matrix A $\in \mathbb{R}^{n\times n}$ with rank n is a basis for $\mathbb{R}^n$. Is every basis equally useful? Certainly they all span the space; however, consider two orthogonal vectors in $\mathbb{R}^2$ versus two linearly independent vectors with a small angle between them. While any vector in $\mathbb{R}^2$ can be constructed from either basis, the basis consisting of orthogonal vectors will require smaller coefficients to construct most vectors. This difference can quickly grow large when constructing many vectors or vectors at many angles. Thus, we'd prefer to have orthogonal column vectors in our spanning matrix.

We thus aim to factor a matrix A into the product QX, where Q is an orthogonal matrix. We consider what it means for a matrix to be orthogonal. Recall that $x, y \in \mathbb{R}^n$ are orthogonal if and only if $\langle x, y\rangle = 0$.

Definition 3.1. *A collection of vectors $\{x_1, \dots, x_m\} \subset \mathbb{R}^n$ is:*

(1) ***orthogonal*** *if $\langle x_i, x_j\rangle = 0$ for all $i \neq j$.*
(2) ***orthonormal*** *if, in addition, $|x_i| = \sqrt{\langle x_i, x_i\rangle} = 1$.*

Note that if $\{x_1, \dots, x_m\} \subset \mathbb{R}^n$ is orthogonal, then it can easily be transformed in to an orthonormal set: $\left\{\frac{x_1}{|x_1|}, \dots, \frac{x_m}{|x_m|}\right\} \subset \mathbb{R}^n$.

Definition 3.2. *A square matrix $Q \in \mathbb{R}^{nxn}$ is orthogonal if $Q^TQ = QQ^T = I$. (i.e. $Q^{-1} = Q^T$).*

Note that for an orthogonal matrix, $\{Q_{:i}\}_{i=1}^n$ is an orthonormal base for $\mathbb{R}^n$ and so is $\{Q_{i:}\}_{i=1}^n$.

Proposition 3.3. *Suppose $Q \in \mathbb{R}^{m\times n}$ has orthogonal columns. Then $|Qx| = |x|$, for every $x \in \mathbb{R}^n$.*

Proof. We have

$$|Qx|^2 = \langle Qx, Qx\rangle = \langle x, Q^TQx\rangle = \langle x, x\rangle = |x|^2.$$

An alternative proof without use of inner product notation:

$$|Qx|^2 = (Qx)^TQx = x^TQ^TQx = x^Tx = x\cdot x = |x|^2.$$

□

We will make significant use of the notion of **one sided orthogonality**. If $Q \in \mathbb{R}^{m\times n}$ with $m > n$, then Q can have orthonormal columns ($Q^TQ = I$), but not orthonormal rows (since the number of rows is larger than the number of columns, Q must have linearly *dependent* rows). On the other hand, if $Q \in \mathbb{R}^{m\times n}$ with $m < n$, then Q can have orthonormal rows ($QQ^T = I$), but not orthonormal columns (since the number of columns is larger than the number of rows, Q must have linearly *dependent* columns).

Theorem 3.4. *Every subspace of $\mathbb{R}^n$ has an orthonormal basis.*

Proof. There exists a basis $X := \{x_1, \dots, x_k\}$ of the subspace. The Gram-Schmidt procedure will yield an orthonormal system $\widehat{X} := \{\widehat{x_1}, \dots, \widehat{x_k}\}$ such that $\text{span}(\widehat{X}) = \text{span}(X)$. To define $\widehat{X}$, we first define Y as follows:

$$y_1 := x_1, \qquad y_2 := x_2 - \frac{\langle x_2, y_1\rangle}{|y_1|^2}y_1, \qquad \dots \qquad y_k := x_k - \sum_{j=1}^{k-1}\frac{\langle x_k, y_j\rangle}{|y_j|^2}y_j.$$

Note that the fractional terms represent the orthogonal projections of y_j onto x_k.

Now we claim that $\text{span}(Y) = \text{span}(X)$, which we will show by induction on the index t. By definition, $\text{span}(y_1) = \text{span}(x_1)$. Now suppose that $\text{span}(y_1, \dots, y_t) = \text{span}(x_1, \dots, x_t)$ for some $1 < t < k$. Note that $y_{t+1} = x_{t+1} - \sum_{j=1}^{t} \alpha_j y_j$ for some scalars $\{\alpha_j\}_{j=1}^{t}$. The summation term is in $\text{span}(y_1, \dots, y_t)$ which is $\text{span}(x_1, \dots, x_t)$ by the induction hypothesis. Thus we have that $y_{t+1} = x_{t+1} - \sum_{j=1}^{t} \beta_j x_j$ for some scalars $\{\beta_j\}_{j=1}^{t}$, whence $y_{t+1} \in \text{span}(x_1, \dots, x_{t+1})$. On the other hand, $x_{t+1} = y_{t+1} - \sum_{j=1}^{t} \alpha_j y_j \in \text{span}(y_1, \dots, y_{t+1})$, so $\text{span}(x_1, \dots, x_{t+1}) = \text{span}(y_1, \dots, y_{t+1})$, which is the desired conclusion.

Next we claim that Y is an orthogonal set, which we again show by induction. Note that

$$\begin{aligned}
\langle y_1, y_2 \rangle &= \left\langle x_1, x_2 - \frac{\langle x_2, x_1 \rangle}{|x_1|^2} x_1 \right\rangle \\
&= \langle x_1, x_2 \rangle - \frac{\langle x_2, x_1 \rangle}{|x_1|^2} \langle x_1, x_1 \rangle \\
&= \langle x_1, x_2 \rangle - \langle x_1, x_2 \rangle
\end{aligned}$$

by symmetry of the inner product and the fact that $\langle x_1, x_1 \rangle = |x_1|^2$. By induction, suppose that $\langle y_i, y_j \rangle = 0$ for all $1 \le i < j \le t$ for some t. Then consider any $i < t+1$, and we have

$$\begin{aligned}
\langle y_i, y_{t+1} \rangle &= \left\langle y_i, x_{t+1} - \sum_{j=1}^{t} \frac{\langle x_{t+1}, y_j \rangle}{|y_j|^2} y_j \right\rangle \\
&= \langle y_j, x_{t+1} \rangle - \sum_{j=1}^{t} \frac{\langle x_{t+1}, y_j \rangle}{|y_j|^2} \langle y_i, y_j \rangle \\
&= \langle y_j, x_{t+1} \rangle - \frac{\langle x_{t+1}, y_i \rangle}{|y_i|^2} \langle y_i, y_i \rangle \\
&= 0.
\end{aligned}$$

The first equality is by definition; the second is by linearity of the inner product; the third is by the induction hypothesis, and the last follows the same reasoning as the $t = 2$ step.

Thus Y is an orthogonal basis for the subspace, so defining $\widehat{x}_j := \frac{y_j}{|y_j|}$ yields an orthonormal basis. □

3.3. **Orthogonal Projections.** Suppose $Q^T Q = I$, and let $P = QQ^T$. Then

$$P^2 = QQ^T QQ^T = QIQ^T = QQ^T = P.$$

Such a matrix is called a **projection**.

Definition 3.5. *If $P^2 = P = P^T$, then P is called an **orthogonal projection** (sometimes called an **orthogonal projector**) onto* $\text{Col}(P)$.

Proposition 3.6. *If P is an orthogonal projection, then for every $x \in \mathbb{R}^n$, Px is the unique solution to*

$$\min_{y \in \text{Col}(P)} |x - y|.$$

Definition 3.7. *Subspaces $\mathscr{S}_1, \mathscr{S}_2 \subset \mathbb{R}^n$ are orthogonal if $\langle x, y \rangle = 0$ for all $x \in \mathscr{S}_1$ and $y \in \mathscr{S}_2$.*

Note that $\mathscr{S}_2$ does not have to equal $\mathscr{S}_1^{\perp}$ for orthogonality to hold (for example, two lines on the xy–plane in $\mathbb{R}^3$ are orthogonal subspaces, but are not orthogonal complements.

4. Lecture 4: The Spectral Decomposition

The goal of this section is to prove the Spectral Theorem, stated as follows.

Theorem 4.1 (Spectral Theorem)**.** *Let $A \in \mathbb{R}^{n\times n}$ be symmetric ($A^T = A$). Then there exists an orthonormal basis of eigenvectors of A, $\{v_1, \dots, v_n\}$, such that*

$$A = V\Lambda V^{-1} = V\Lambda V^T,$$

where $V = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix}$ and Λ is diagonal and contains the eigenvectors of A.

Let's begin with a reminder of the following fundamental concept of Linear Algebra.

Definition 4.2. *Let $A \in \mathbb{R}^{n\times n}$. A scalar–vector pair $(\lambda, v) \in \mathbb{R} \times \mathbb{R}^n$ with $v \neq 0$ is an **eigenvalue/eigenvector pair** of A if*

$$Av = \lambda v.$$

Geometrically, eigenvectors are ones for which when they are multiplied by A, the product does not change direction (except for possibly a backward directional change), but rather the eigenvector is stretched in its same direction by some scalar factor. First, let's see how we can compute eigenvalue/eigenvector pairs for a given matrix. To start, we need the definition and properties of determinants.

Definition 4.3. *Given a 2×2 matrix $A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$, its **determinant** is*

$$\det(A) := ad - bc.$$

For a general $A \in \mathbb{R}^{n\times n}$, its determinant is defined iteratively as

$$\det(A) = \sum_{k=1}^{n} (-1)^k a_{1k} \det(M_k)$$

*where the **minor**, M_k, is the $(n-1)\times(n-1)$ matrix formed by deleting the first row and k–th column of A.*

Note that $\det(A)$ is always a scalar, and has the following important properties.

Theorem 4.4. *Let $A, B \in \mathbb{R}^{n\times n}$. Then the following hold:*

(1) $\det(I) = 1$
(2) $\det(A^T) = \det(A)$
(3) $\det(AB) = \det(A)\det(B)$
(4) $\det(A^{-1}) = \frac{1}{\det(A)}$ *provided A is invertible*
(5) $\det(\alpha A) = \alpha^n \det(A)$ *for any $\alpha \in \mathbb{R}$*
(6) If A has eigenvalues $\lambda_1, \dots, \lambda_n$, then $\det(A) = \prod_{i=1}^n \lambda_i$
(7) $\det(A) \neq 0$ *if and only if A is invertible.*

Now eigenvalue/eigenvector pairs are distinctly related to determinants in the following way.

Proposition 4.5. *Let $A \in \mathbb{R}^{n\times n}$. Then $(\lambda, v) \in \mathbb{R} \times \mathbb{R}^n$ with $v \neq 0$ is an eigenvalue/eigenvector pair of A if and only if*

$$\det(A - \lambda I) = 0.$$

Proof. For the forward direction, suppose that $Av = \lambda v$. Then $(A - \lambda I)v = 0$, which means that $v \in \mathcal{N}(A - \lambda I)$. Since $v \neq 0$, $A - \lambda I$ has nontrivial nullspace, and hence is not invertible; so by Theorem 4.4(7), $\det(A - \lambda I) = 0$.

To see the converse, if $\det(A-\lambda I)=0$, then A is not invertible, and hence has non-trivial nullspace. Thus there exists a nonzero v such that $(A-\lambda I)v=0$, which implies that $Av=\lambda v$. □

Something important to note is that even if a matrix is real-valued, its eigenvalues can be complex. Indeed, if

$$A := \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

then the solution to $\det(A-\lambda I)=\lambda^2+1=0$ is $\lambda=\pm i$ where i is the imaginary unit. Nonetheless, an important class of matrices do in fact have real eigenvalues as the following proposition shows.

Proposition 4.6. *Let $A\in\mathbb{R}^{n\times n}$ be symmetric (i.e., $A^T=A$). Then all eigenvalues of A are real.*

Proof. Suppose (λ, v) is an eigenvalue/eigenvector pair of A. By conjugate-linearity in the second argument for inner products, we have

$$\lambda\langle v,v\rangle = \langle \lambda v, v\rangle = \langle Av, v\rangle = \left\langle v, A^T v\right\rangle = \langle v, Av\rangle = \langle v, \lambda v\rangle = \overline{\lambda}\langle v, v\rangle . \tag{2}$$

Since $v\neq 0$, $\langle v,v\rangle\neq 0$, thus (2) implies that $\lambda=\overline{\lambda}$, hence λ must be real. □

Next we will note that distinct eigenvalues yield orthogonal eigenvectors.

Proposition 4.7. *Suppose $A\in\mathbb{R}^{n\times n}$ is symmetric, and (λ_1, v_1) and (λ_2, v_2) are eigenvalue/eigenvector pairs of $A\in\mathbb{R}^{n\times n}$. If $\lambda_1\neq\lambda_2$, then v_1 is orthogonal to v_2.*

Proof. Note that (since $\lambda_1,\lambda_2\in\mathbb{R}$)

$$\lambda_1\langle v_1, v_2\rangle = \langle \lambda_1 v_1, v_2\rangle = \langle Av_1, v_2\rangle = \langle v_1, Av_2\rangle = \langle v_1, \lambda_2 v_2\rangle = \lambda_2\langle v_1, v_2\rangle .$$

This implies that $(\lambda_1-\lambda_2)\langle v_1, v_2\rangle = 0$, but $\lambda_1-\lambda_2\neq 0$ by assumption, so v_1 and v_2 must be orthogonal. □

This leads to the following observation:

Proposition 4.8 (Easy case of the Spectral Theorem)**.** *Suppose that $A\in\mathbb{R}^{n\times n}$ is symmetric and has n distinct, nonzero eigenvalues, $\lambda_1,\dots,\lambda_n$. Then A has n orthonormal eigenvectors, and we may write*

$$A = S\Lambda S^{-1} = S\Lambda S^T = \sum_{i=1}^{n}\lambda_i S_{:i}S_{:i}^T$$

where the columns of S are the eigenvectors of A and $\Lambda=\operatorname{diag}(\lambda_1,\dots,\lambda_n)$.

Note that this result proves the Spectral Theorem in the case that A has distinct eigenvalues, so the only case we have left to consider is that when A has repeated eigenvalues. We will illustrate a couple of ways one can finish the proof of the general case.

4.1. **First Proof.** Our first argument for proving the Spectral Theorem is a more constructive one which boils down to repeatedly doing Gram–Schmidt on blocks of the initial matrix.

Suppose that (λ_1, v_1) is any eigenvalue/eigenvector pair of A and note that without loss of generality we may assume that v_1 is a unit vector (if not divide by its norm). We know that there is an orthonormal basis for $\mathbb{R}^n$ containing v_1, say $\{v_1, w_2,\dots,w_n\}$. Let

Q_1 be the matrix with these basis vectors as columns. Then by orthonormality of its columns, we have

$$Q_1 A Q_1^T = \begin{bmatrix} \lambda_1 & 0 \\ 0 & B_1 \end{bmatrix}$$

for some symmetric matrix $B_1 \in \mathbb{R}^{(n-1)\times(n-1)}$.

Before proceeding we need the following crucial observation

Lemma 4.9. *Let $A \in \mathbb{R}^{n\times n}$. If $Q \in \mathbb{R}^{n\times n}$ is an orthogonal matrix, then QAQ^T has the same eigenvalues as A.*

Proof. Let (λ, v) be an eigenvalue/eigenvector pair of QAQ^T. Then $QAQ^T v = \lambda v$, and by multiplying by Q^T on the left, we see that

$$AQ^T v = \lambda Q^T v,$$

which means that $(\lambda, Q^T v)$ is an eigenvalue/eigenvector pair of A. Since both A and QAQ^T have the same rank and this argument works for all eigenpairs, the claim is proven. □

Now we may apply the conclusion of Lemma 4.9 to note that $Q_1 A Q_1^T$ has the same eigenvalues as A, hence the eigenvalues of B_1 are $\lambda_2, \dots, \lambda_n$. Now we iterate the same argument as above to find an orthogonal $\widetilde{Q}$ for which

$$\widetilde{Q} B_1 \widetilde{Q}^T = \begin{bmatrix} \lambda_2 & 0 \\ 0 & B_2 \end{bmatrix},$$

and we set

$$Q_2 := \begin{bmatrix} 1 & 0 \\ 0 & \widetilde{Q} \end{bmatrix}.$$

Now we have that

$$Q_2 Q_1 A Q_1^T Q_2^T = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & B_2 \end{bmatrix}.$$

Iterating this procedure, we find $Q_1, \dots, Q_n$ which diagonalize A into $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ as desired, and we take $S := Q_n \dots Q_1$ to get the desired decomposition.

4.2. **Second Proof.** Our second argument is nonconstructive, but follows from somewhat more basic Linear Algebra facts. We begin with the following observation.

Lemma 4.10. *Let $A \in \mathbb{R}^{n\times n}$ be symmetric, and let $\mathcal{S}$ be any subspace of $\mathbb{R}^n$. If $Ax \in \mathcal{S}$ for every $x \in \mathcal{S}$, then $Ay \in \mathcal{S}^\perp$ for every $y \in \mathcal{S}^\perp$.*

Proof. By assumption, $\langle Ax, y\rangle = 0$, but by symmetry, $\langle Ax, y\rangle = \langle x, Ay\rangle$, so $Ay \in \mathcal{S}^\perp$ since x is an arbitrary element of $\mathcal{S}$. □

Lemma 4.11. *If $A \in \mathbb{R}^{n\times n}$ is symmetric and $\mathcal{S}$ is a nonzero subspace of $\mathbb{R}^n$ such that $Ax \in \mathcal{S}$ for every $x \in \mathcal{S}$, then $\mathcal{S}$ contains an eigenvector of A.*

Proof. First of all, we know that $\mathcal{S}$ contains an orthonormal basis by Theorem 3.4; call it $\{u_1, \dots, u_k\}$. Since $Au_j \in \mathcal{S}$ for all j, there are scalars $\{r_{ij}\}_{i,j=1}^k$ such that

$$Au_j = \sum_{i=1}^{k} r_{ij} u_i.$$

Indeed $r_{ij} = \langle Au_j, u_i\rangle$. Let R be the $k \times k$ matrix with entries $R_{ij} = r_{ij}$. Then note that R is symmetric because $r_{ji} = \langle Au_i, u_j\rangle = \langle u_i, Au_j\rangle = \langle Au_j, u_i\rangle = r_{ij}$.

Now suppose that (λ, v) is an eigenvalue/eigenvector pair of R. Then define $w := \sum_{j=1}^{k} v_j u_j$ where v_j is the j–th component of the vector v, and u_j is still one of the basis vectors for $\mathscr{S}$. Then we have

$$\begin{aligned} Aw &= \sum_{j=1}^{k} v_j A u_j \\ &= \sum_{j=1}^{k} v_j \sum_{i=1}^{k} r_{ij} u_i \\ &= \sum_{i=1}^{k} \left(\sum_{j=1}^{k} r_{ij} v_j \right) u_i \\ &= \sum_{i=1}^{k} (Rv)_i u_i \\ &= \sum_{i=1}^{k} \lambda v_i u_i \\ &= \lambda w. \end{aligned}$$

Hence (λ, w) is an eigenvalue/eigenvector pair of A. □

Now we are ready to complete the proof of the Spectral Theorem.

Proof of the Spectral Theorem. Let (λ_1, v_1) be an eigenvalue/eigenvector pair of A, and set $\mathscr{S}_1 := \text{span}(v_1)$. Note that $A(\alpha v_1) = \lambda \alpha v_1 \in \mathscr{S}_1$, so by Lemma 4.10, $Ay \in \mathscr{S}_1^{\perp}$ for every $y \in \mathscr{S}_1^{\perp}$. If $\mathscr{S}_1^{\perp} \neq \{0\}$, then Lemma 4.11 implies that there exists an eigenvector $v_2 \in \mathscr{S}_1^{\perp}$ (which is necessarily orthogonal to v_1). Now if $n > 2$, then set $\mathscr{S}_2 := \text{span}(v_1, v_2)$, and note that $Ax \in \mathscr{S}_2$ for every $x \in \mathscr{S}^2$ by a similar argument to the first case. Again apply Lemmas 4.10 and 4.11 to get an orthogonal eigenvector $v_3 \in \mathscr{S}_2^{\perp}$. Continue in this manner to choose $v_1, \ldots, v_n$ and the proof is complete. □

Now let us note that the Spectral Theorem gives us a representation of a matrix in terms of the sum of rank 1 matrices.

Corollary 4.12. *Let $A \in \mathbb{R}^{n \times n}$ be symmetric, and let $\{v_i\}_{i=1}^n$ be an orthonormal basis of eigenvectors of A with corresponding eigenvalues $\{\lambda_i\}_{i=1}^n$. Then*

$$A = \sum_{i=1}^{n} \lambda_i v_i v_i^T.$$

Proof. First note that

$$V\Lambda = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \lambda_n \end{bmatrix}$$

$$= \begin{bmatrix} \lambda v_1 & \cdots & \lambda_n v_n \end{bmatrix}.$$

Since $V^{-1} = V^T$, we have

$$V\Lambda V^{-1} = \begin{bmatrix} \lambda v_1 & \cdots & \lambda_n v_n \end{bmatrix} \begin{bmatrix} {v_1}^T \\ \vdots \\ {v_n}^T \end{bmatrix} = \sum_{i=1}^{n} \lambda_i v_i {v_i}^T$$

as required. □

5. Lectures 5 & 6: The Singular Value Decomposition

The goal of these lectures is to prove the Singular Value Decomposition (SVD), which is the analogue of the Spectral Theorem we saw last time for arbitrary matrices (i.e., they can be rectangular, and no assumption is made on their rank or other structure such as symmetry). Here is the full statement of the theorem we want to prove.

Theorem 5.1 (Singular Value Decomposition)**.** *Let $A \in \mathbb{R}^{m\times n}$. There exist orthogonal matrices $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ and a matrix $\Sigma \in \mathbb{R}^{m\times n}$ with nonnegative entries only along its main diagonal such that*

$$A = U\Sigma V^T = \sum_{i=1}^{\rho} \sigma_i U_{:i} V_{:i}^T,$$

where $\rho = \min\{m,n\}$. Moreover, we may take $\rho = \operatorname{rank}(A)$.

To begin, we need one more ingredient about eigenvalues of symmetric matrices.

Definition 5.2. *A symmetric matrix $A \in \mathbb{R}^{n\times n}$ is **positive definite** provided $\langle Ax, x\rangle > 0$ for all $x \in \mathbb{R}^n$. If instead $\langle Ax, x\rangle \geq 0$ for all $x \in \mathbb{R}^n$, the matrix is called **positive semi-definite**. For short, we will call symmetric, positive definite (respectively, symmetric positive semi-definite) matrices SPD (respectively, SPSD) for short.*

Proposition 5.3. *Any matrix of the form AA^T or A^TA is SPSD.*

Proof. Simply note that $\langle A^TAx, x\rangle = \langle Ax, Ax\rangle = |Ax|^2 \geq 0$ (this could be 0 if A has non-trivial nullspace). For AA^T, simply note this is of the form B^TB where $B = A^T$ and the same argument works. □

Proposition 5.4. *If $A \in \mathbb{R}^{n\times n}$ is SPSD, then its eigenvalues are real and nonnegative.*

Proof. That the eigenvalues are real follows from Proposition 4.6. To see positivity, suppose that (λ, v) is an eigenvalue/eigenvector pair of A. Then $\lambda\langle v, v\rangle = \langle \lambda v, v\rangle = \langle Av, v\rangle$, whence because $v \neq 0$,

$$\lambda = \frac{\langle Av, v\rangle}{\langle v, v\rangle} \geq 0$$

where the inequality follows from the fact that A is positive semi-definite. □

5.1. **Proof of the SVD.** To begin, consider the $A^TA \in \mathbb{R}^{nxn}$, which is SPSD by Proposition 5.3, and hence has real, nonnegative eigenvalues (by Proposition 5.4, say $\lambda_1, ..., \lambda_k$, with corresponding orthonormal eigenvectors $v_1, ..., v_k$.

Now we note that A^TA and AA^T have the same eigenvalues. More precisely, the following holds.

Lemma 5.5. *Let $A \in \mathbb{R}^{m\times n}$ be arbitrary. If (λ, v) is an eigenvalue/eigenvector pair of A^TA, then (λ, Av) is an eigenvalue/eigenvector pair of AA^T.*

Proof. By assumption, $A^TAv = \lambda v$, whereupon multiplication by A yields $(AA^T)Av = \lambda(Av)$. Since 0 cannot be an eigenvector, we must confirm that v is not in the null space of A, but thisfollows from the fact that $\mathcal{N}(A) = \mathcal{N}(A^TA)$. □

Now, since we have the eigenpairs of A^TA in hand, Lemma 5.5 suggests that we set

$$u_i := \frac{Av_i}{\sqrt{\lambda_i}}, \quad i = 1, \ldots, k. \tag{3}$$

Lemma 5.5 implies that $u_1, ..., u_k$ are still eigenvectors of AA^T with associated eigenvalues $\lambda_1, ..., \lambda_k$. Now we claim that $\{u_i\}_{i=1}^k$ is an orthonormal set.

Lemma 5.6. *Let $\{u_i\}_{i=1}^k$ be defined as in (3). Then $\{u_i\}_{i=1}^k$ is an orthonormal set.*

Proof. Let i and j be fixed. By definition,

$$\langle u_i, u_j \rangle = \left\langle \frac{1}{\sqrt{\lambda_i}} A v_i, \frac{1}{\sqrt{\lambda_j}} A v_j \right\rangle = \frac{1}{\sqrt{\lambda_i \lambda_j}} \langle A v_i, A v_j \rangle = \frac{1}{\sqrt{\lambda_i \lambda_j}} \langle v_i, A^T A v_j \rangle.$$

Since (λ_j, v_j) is an eigenvalue/eigenvector pair for $A^T A$,

$$\langle u_i, u_j \rangle = \frac{1}{\sqrt{\lambda_i \lambda_j}} \langle v_i, \lambda_j v_j \rangle = \frac{\sqrt{\lambda_j}}{\sqrt{\lambda_i}} \langle v_i, v_j \rangle = \begin{cases} 0, & i \neq j \\ 1, & i = j. \end{cases}$$

The final calculation follows from the fact that $\{v_i\}_{i=1}^k$ is orthonormal, and the fact that $\{u_i\}_{i=1}^k$ is orthonormal follows directly from the above calculation. □

Now letting $\sigma_i = \sqrt{\lambda_i}$ for $i = 1, \ldots, k$, define

$$U_k := \begin{bmatrix} | & & | \\ u_1 & \cdots & u_k \\ | & & | \end{bmatrix}, \quad V_k := \begin{bmatrix} | & & | \\ v_1 & \cdots & v_k \\ | & & | \end{bmatrix}, \quad \Sigma_k := \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & \sigma_k \end{bmatrix}.$$

Lemma 5.7. *With the above definitions, we have $U_k^T A V_k = \Sigma_k$.*

Proof. Note that by the inner product definition of matrix multiplication,

$$U_k^T A V_k = \begin{bmatrix} \langle Av_1, u_1 \rangle & \langle Av_2, u_1 \rangle & \cdots & \langle Av_k, u_1 \rangle \\ \langle Av_1, u_2 \rangle & \langle Av_2, u_2 \rangle & \cdots & \langle Av_k, u_2 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle Av_1, u_k \rangle & \langle Av_2, u_k \rangle & \cdots & \langle Av_k, u_k \rangle \end{bmatrix}.$$

Now by definition of u_i and σ_i, we have

$$\langle Av_i, u_j \rangle = \left\langle Av_i, \frac{Av_j}{\sigma_j} \right\rangle = \frac{1}{\sigma_j} \langle Av_i, Av_j \rangle = \frac{1}{\sigma_j} \langle v_i, A^T A v_j \rangle = \frac{\lambda_i^2}{\sigma_j} \langle v_i, v_j \rangle$$

Thus by orthonormality of $\{v_i\}$, we have

$$\langle Av_i, u_j \rangle = \begin{cases} 0, & i \neq j \\ \sigma_i, & i = j. \end{cases} \tag{4}$$

The desired conclusion follows from (4). □

Proof of the SVD. Our proof of the SVD is almost complete. First, we complete $\{u_1, \ldots, u_k\}$ to an orthonormal basis of $\mathbb{R}^{m \times m}$, denoted $\{u_1, \ldots, u_m\}$, and likewise complete $\{v_1, \ldots, v_k\}$ to an orthonormal basis of $\mathbb{R}^{n \times n}$, say $\{v_1, \ldots, v_n\}$. We also make Σ an $m \times n$ matrix by adding zeros to make up the rest of the entries.

Our final claim is that $U^T A V = \Sigma$. To see this, first note that $v_{k+1}, \ldots, v_n \in \mathcal{N}(A)$. This implies that if $i > k$, then the i, j–th entry of $U^T A V$ is $\langle Av_i, u_j \rangle = 0$. For $1 \leq i \leq k$ and $1 \leq j \leq k$, we have already seen that $\langle Av_i, u_j \rangle$ is either 0 or σ_i by (4). We only need to check that for $1 \leq i \leq k$ and $j > k$ that $\langle Av_i, u_j \rangle = 0$ to verify the claim. In this case, we go back to the definition of u_j and use the same argument as in Lemma 5.7 to see that

$$\langle Av_i, u_j \rangle = \langle \sigma_i u_i, u_j \rangle = 0.$$

The conclusion follows that $U^T AV = \Sigma$.

The following calculation, which follows by orthogonality of the matrices U and V, finishes the proof:

$$U\Sigma V^T = UU^T AVV^T = A.$$

□

5.2. **Understanding the SVD.** By the same calculation as in the proof of Corollary 4.12, we find that if $A = U\Sigma V^T$, then

$$A = \sum_{i=1}^{\text{rank}(A)} \sigma_i u_i v_i^T. \tag{5}$$

As a matter of terminology, we call the columns of U the **left singular vectors** of A, the columns of V the **right singular vectors** of A, and $\sigma_1, \ldots, \sigma_{\text{rank}(A)}$ the **singular values** of A. Additionally, we assume the convention that the singular values are in descending order so that $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\text{rank}(A)} \geq 0$.

Now let's take a step back and consider what the SVD is really telling us. Let us consider the geometry of a linear transformation (matrix) $A : \mathbb{R}^2 \to \mathbb{R}^2$ as shown in Figure 2.

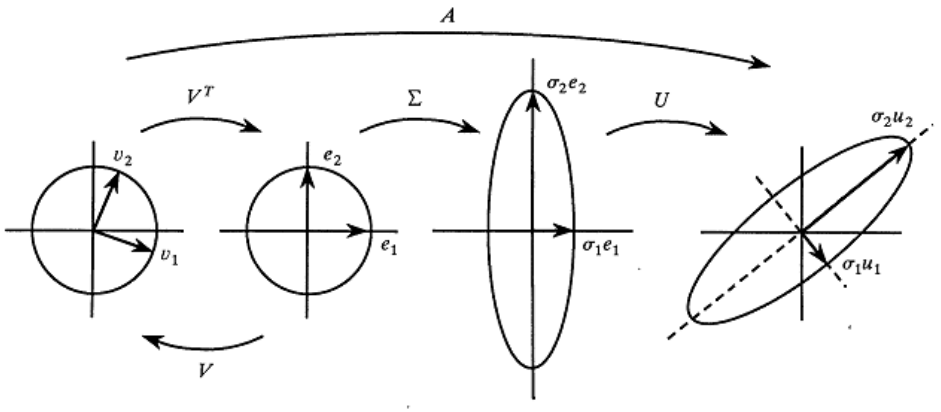

FIGURE 2. Geometric representation of the SVD. Figure from https://blogs.sas.com/content/iml/2017/08/28/singular-value-decomposition-svd-sas.html

Figure 2 illustrates the fact that V^T maps the generic orthonormal basis $\{v_1, v_2\}$ onto the canonical basis $\{e_1, e_2\}$. Then Σ stretches e_1 and e_2, and U is another rotation onto a new orthonormal basis. Hence the SVD for square matrices can be seen as the sequence: rotate, stretch, rotate again. In higher dimensions, rotation means multiplication by orthogonal matrices with determinant 1.

Another thing to note about what the SVD is doing algebraically is that it finds two orthogonal bases in which the transformation A mapping the basis $\{v_1, v_2\}$ to $\{u_1, u_2\}$ is diagonal. In particular, we have that $Av_i = \sigma_i u_i$.

6. Lecture 7: The Eckhart–Young–Mirsky Theorem

The goal of this lecture is to show the conclusion of the Eckhart–Young–Mirsky Theorem, which says that in a certain sense that will be made precise in what follows, the matrix $A_k := U_k \Sigma_k V_k^T$ is the "best" rank k approximation of A. For the moment, we must content ourselves with a discussion of norms to better clarify this statement.

Definition 6.1. *A function $\|\cdot\| : \mathbb{R}^n \to \mathbb{R}$ is a **norm** if*

(1) $\|x\| \geq 0$, and $\|x\| = 0$ if and only if $x = 0$
(2) $\|\alpha x\| = |\alpha| \|x\|$ for all $\alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$
(3) $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$.

The inequality in Definition 6.1(3) is called the Triangle Inequality.

Definition 6.2. *The vector ℓ_p norms on $\mathbb{R}^n$ are defined by*

$$\|x\|_p := \left(\sum_{i=1}^{n} |x_i|^p \right)^{\frac{1}{p}}, \qquad 1 \leq p < \infty,$$

and

$$\|x\|_\infty := \max_{1 \leq i \leq n} |x_i|.$$

The case $p = 2$ is typically called the *Euclidean norm* on $\mathbb{R}^n$.

Example 6.3. Let $x = \begin{bmatrix} -2 & 1 & -1 \end{bmatrix}^T$. Then

$$\|x\|_2 = \sqrt{4 + 1 + 1} = \sqrt{6}, \quad \|x\|_1 = 2 + 1 + 1 = 4, \quad \|x\|_\infty = \max\{2, 1, 1\} = 2.$$

For illustration, let's consider what the unit balls in each of these norms look like (at least in $\mathbb{R}^2$).

Definition 6.4. *For given $1 \leq p \leq \infty$ and $n \in \mathbb{N}$, the **unit ball** in $\mathbb{R}^n$ in the ℓp norm is the set*

$$B_p^n := \{x \in \mathbb{R}^n : \|x\|_p \leq 1\}.$$

Figure 3 illustrates the shape of the unit balls in $\mathbb{R}^2$ for the ℓ_1, ℓ_2, and ℓ_∞ norms. Note that for $1 < q < 2 < r < \infty$, we have that

$$B_1^n \subset B_q^n \subset B_2^n \subset B_r^n \subset B_\infty^n.$$

This fact comes with the relationship that for any $x \in \mathbb{R}^n$,

$$\|x\|_\infty \leq \|x\|_r \leq \|x\|_2 \leq \|x\|_q \leq \|x\|_1.$$

6.1. **Induced Matrix Norms.** Given a norm, $\|\cdot\|_a$ which is well-defined on both $\mathbb{R}^m$ and $\mathbb{R}^n$, the corresponding induced matrix norm is the function $\|\cdot\|_a : \mathbb{R}^{m \times n} \to \mathbb{R}$ given by

$$\|A\|_a := \max_{x \in \mathbb{R}^n \setminus \{0\}} \frac{\|Ax\|_a}{\|x\|_a}.$$

Noticing that the term above is $\max_{x \in \mathbb{R}^n \setminus \{0\}} \left\| A\left(\frac{x}{\|x\|_a} \right) \right\|_a$, we have that

$$\|A\|_a = \max_{x \in \mathbb{R}^n, \|x\|_a = 1} \|Ax\|_a.$$

One important thing to note is that not all matrix norms are induced by a vector norm. For example, the **Frobenius norm** defined by

$$\|A\|_F := \left(\sum_{i=1}^{m} \sum_{j=1}^{n} |A_{ij}|^2 \right)$$

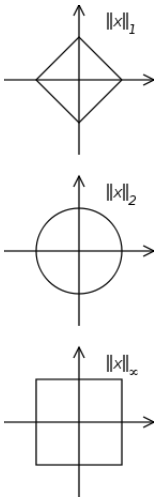

FIGURE 3. Unit Ball in ℓp -norms for $p = 1, 2, \infty$
Figure from https://upload.wikimedia.org/wikipedia/commons/thumb/4/4d/Vector_norms.svg/140px-Vector_norms.svg.png

is not an induced matrix norm.

One of the most important norms we will consider is the **Spectral Norm** defined by

$$\|A\|_2 := \max_{\|x\|_2=1} \|Ax\|_2.$$

It is of interest to note that while they are not defined in this way, we may relate both the Frobenius and Spectral norms with the singular values of A in the following way.

Theorem 6.5. *For $A \in \mathbb{R}^{m\times n}$, the following hold:*

(1) $\|A\|_2 = \sigma_1(A)$

(2) $\|A\|_F = \left(\sum_{i=1}^{\text{rank}(A)} \sigma_i(A)^2\right)^{\frac{1}{2}}$.

Theorem 6.5 suggest the definition of more general norms defined by the ℓ_p norms of the vectors of singular values of A:

Definition 6.6. *For any $1 \le p \le \infty$, the **Schatten** p**–norm** is given by*

$$\|A\|_{Sp} := \|(\sigma_1(A), \ldots, \sigma_{\text{rank}(A)}(A))\|_p = \left(\sum_{i=1}^{rank(A)} (\sigma_i)^p\right)^{\frac{1}{p}}.$$

Proposition 6.7. *Schatten p–norms are,*

(1) ***Submultiplicative****:* $\|AB\|_{S_p} \le \|A\|_{S_p}\|B\|_{S_p}$

(2) ***Unitarily Invariant****:* $\|QAW\|_{S_p} = \|A\|_{S_p}$ *for all orthogonal Q and W.*

Proof. Unitary invariance follows by the fact that singular values don't change after multiplication by orthogonal matrices. In particular, $\|A\|_{S_p} = \|\Sigma\|_{S_p} = \|\{\sigma_i(A)\}\|_p$. We will not give the proof of submultiplicativity here. □

Corollary 6.8. *If $\{u_i\}_{i=1}^r, \{v_i\}_{i=1}^r$ are orthonormal systems, then*

$$\left\| \sum_{i=1}^{r} \sigma_i u_i v_i^T \right\|_{S_p} = \|(\sigma_1, ..., \sigma_r)\|_p = \left(\sum_{i=1}^{r} |\sigma_i|^p \right)^{\frac{1}{p}}.$$

6.2. **The Eckhart–Young–Mirsky Theorem.** We now have enough definitions at our disposal to state the main theorem of this lecture, which is the following.

Theorem 6.9 (Eckhart-Young-Mirsky)**.** *For every $A \in \mathbb{R}^{mxn}$ with $A = U\Sigma V^T$, $A_k = U_k \Sigma_k V_k^T$ is a solution to*

$$\min_{\substack{B \in \mathbb{R}^{m \times n} \\ \text{rank}(B) \leq k}} \|A - B\|_{S_p}$$

for all $1 \leq p \leq \infty$.

Note that this theorem implies that for any B such that $\text{rank}(B) \leq k$ we have that $\|A - U_k \Sigma_k V_k^T\|_{S_p} \leq \|A - B\|_{S_p}$.

Proof for Spectral Norm (S_∞). Without loss of generality assume $n \leq m$ (otherwise the following analysis holds for A^T). First, note that

$$\begin{aligned} \|A - A_k\|_2 &= \left\| \sum_{i=1}^{n} \sigma_i u_i v_i^T - \sum_{i=1}^{k} \sigma_i u_i v_i^T \right\|_2 \\ &= \left\| \sum_{i=k+1}^{n} \sigma_i u_i v_i^T \right\|_2 \\ &= \sigma_{k+1}, \end{aligned}$$

where the last line follows from Corollary 6.8. Now let $B \in \mathbb{R}_{m \times n}$ with $\text{rank}(B) \leq k$. Then $\dim(\mathcal{N}(B)) = n - \text{rank}(B)$ by the Rank–Nullity Theorem. So we see $\dim(\mathcal{N}(B)) \geq n - k$ as $\text{rank}(B) \leq k$. Since $\dim(\text{Col}(V_{k+1})) = k + 1$,

$$\dim(\mathcal{N}(B)) + \dim(\text{Col}(V_{k+1})) \geq n - k + k + 1 = n + 1.$$

This implies the existence of an $x \in \mathcal{N}(B) \cap \text{Col}(V_{k+1})$ with $|x| = 1$. Suppose $x = \sum_{i=1}^{k+1} c_i v_i$ (as $x \in \text{Col}(V_{k+1})$), then we may write,

$$\|A - B\|_2^2 = \max_{\|y\|_2 = 1} \|(A - B)y\|_2^2 \geq \|(A - B)x\|_2^2 = \|Ax\|_2^2$$

where the last equality follows from the fact that $x \in \mathcal{N}(B)$. Now we have

$$\begin{aligned} \|Ax\|_2^2 &= \left\| A \sum_{i=1}^{k+1} c_i v_i \right\|_2^2 \\ &= \left\| \sum_{i=1}^{k+1} c_i A v_i \right\|_2^2 \\ &= \left\| \sum_{i=1}^{k+1} c_i \sigma_i u_i \right\|_2^2 \\ &= \sum_{i=1}^{k+1} c_i^2 \sigma_i^2 \\ &\geq \sigma_{k+1}^2 \sum_{i=1}^{k+1} c_i^2 \end{aligned}$$

$$= \sigma_{k+1}^2 \|x\|_2^2$$
$$= \sigma_{k+1}^2.$$

The second to the last line above follows from the orthonormality of $\{u_i\}$. Putting these observations together, we have that for any B with rank at most k,

$$\|A - B\|_2 \geq \sigma_{k+1} = \|A - A_k\|_2.$$

□

7. Lecture 8: PCA (Principal component analysis)

This lecture begins our section on applications of the SVD to data analysis. We begin with one of the most fundamental methods around: Principal Component Analysis, or PCA. Let us start by giving an example

Example 7.1. Suppose a matrix $A \in \mathbb{R}^{m\times n}$ contains n samples of patients from whom many medical observations are collected resulting in a vector of m features for each patient. For example,

$$A = \begin{array}{c} \begin{array}{ccccc} \text{Patient 1} & \text{Patient 2} & \text{Patient 3} & \cdots & \text{Patient n} \end{array} \\ \left[\begin{array}{ccccc} * & * & * & \cdots & * \\ * & * & * & \cdots & * \\ \vdots & \vdots & \vdots & \cdots & \vdots \\ * & * & * & \cdots & * \end{array}\right] \end{array} \begin{array}{l} \text{Feature 1} \\ \text{Feature 2} \\ \vdots \\ \text{Feature m} \end{array}$$

If m and n are very large, it may be difficult or impractical to analyze the data as a whole. However, a medical researcher wants to be able to make decisions about patients based on the data. For example, perhaps the researcher wants to know which of the features they have collected about the patients most influence whether or not they have a particular disease, or perhaps the researcher wants to use the data to detect different populations within the set of patients to find some meaningful pattern. PCA attempts to find the directions of largest variance within the data set.

PCA begins by considering what is called the *covariance matrix* of the data $\{A_{:i}\}_{i=1}^n$ which will be defined momentarily. To do this, we first center the rows of A as follows:

$$\widehat{A}_{ij} := A_{ij} - u_i, \qquad \text{where} \qquad u_i := \sum_{j=1}^n A_{ij}.$$

Note that the rows of $\widehat{A}$ are unit vectors at this point. The second step of PCA is to compute the covariance matrix S of the newly centered data as follows:

$$S_{ij} := \frac{1}{n-1}(\widehat{A}\widehat{A}^T)_{ij} = \frac{1}{n-1}\langle \widehat{A}_{i:}, \widehat{A}_{j:}\rangle = \frac{1}{n-1}\sum_{k=1}^n (A_{ik}-u_i)(A_{jk}-u_j) =: \text{Covar}(A_{i:}, A_{j:}).$$

Note here that if $i = j$, then

$$S_{ii} = \frac{1}{n-1}\sum_{k=1}^n (A_{ik}-\mu_i)^2 = \text{Var}(A_{i:})$$

using the classical definition of variance. We call the entries S_{ij} the *covariance* between rows $A_{i:}$ and $A_{j:}$. The covariance matrix reflects how correlated two features are. Figure 4 gives an example of perfectly correlated features and uncorrelated features.

The third step in PCA is to take the SVD of the covariance matrix, i.e., $S = U\Sigma U^T$ (note that it has this form since it is symmetric). By the Eckhart–Young–Mirsky Theorem, $U_k\Sigma_k U_k^T$ is the best rank k approximation of S. The columns of U are called the **Principal Coordinates**, or **Principal Components** of the centered data $\widehat{A}$.

For completeness, we write the full PCA algorithm below in Algorithm 1.

7.1. Uses of PCA.

Having seen the algorithm, let us now discuss how we may use PCA to analyze data. We will start with a toy example of regression, but it is not linear regression that we may be used to.

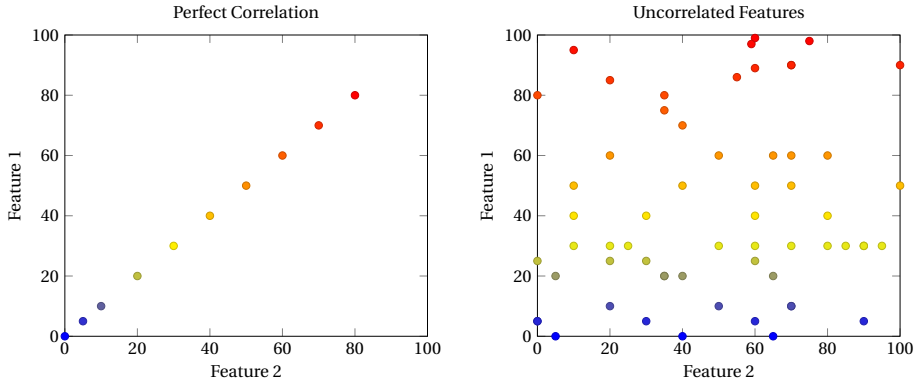

FIGURE 4. An illustration of perfectly correlated features (left) and uncorrelated features (right).

Algorithm 1: Principal Component Analysis (PCA)

Input : Matrix $A \in \mathbb{R}^{m \times n}$ of n observations with m features
Output: Principal components $\{\sigma_1 u_1, \ldots, \sigma_k u_k\}$
for *i=1* **to** n **do**
  $\mu_i = \frac{1}{n} \sum_{j=1}^{n} A_{ij}$
  $\widehat{A}_{i:} = A_{i:} - \mu_i$
end
$S = \frac{1}{n-1} \widehat{A}\widehat{A}^T$
Compute the SVD of S as $S = U\Sigma U^T$
return $\sigma_1 u_1, \ldots, \sigma_k u_k$

Task 1: Orthogonal Regression. Suppose that $A \in \mathbb{R}^{2 \times n}$ corresponds to n point in $\mathbb{R}^2$, and we want to find the line that best passes through them. We could solve a least squares problem (which we will discuss several lectures from now) or we could use PCA. In this case, $S \in \mathbb{R}^{2 \times 2}$, and U consists of 2 columns u_1 and u_2. In this case, the line in the direction of u_1 is the *orthogonal regression line* that minimizes

$$\sum_{i=1}^{n} \text{dist}(A_{i:}, mx + b)^2$$

over all values of m and b. Here the distance function represents the distance from the point $A_{i:}$ to the nearest point on the line $y = mx + b$ (recall that this is the point at which a line passing through $A_{i:}$ meets the line $y = mx + b$ at a right angle). Note that this is different from linear regression, which minimizes the sum of square distances from the y–coordinates of the data points to the line.

Task 2: Capturing Variance in Data. The first principal component of centered data captures the direction in which the largest variance is exhibited. Then the second principal component explains the next direction orthogonal to the first that contains the largest remaining fraction of the variance, and so on. For an illustration of this fact, see Figure 5.

Task 3: Dimensionality Reduction. Oftentimes, data that we collect is high-dimensional, i.e., m is very large. For example, an iPhone 11 takes 12 megapixel images. Therefore, if

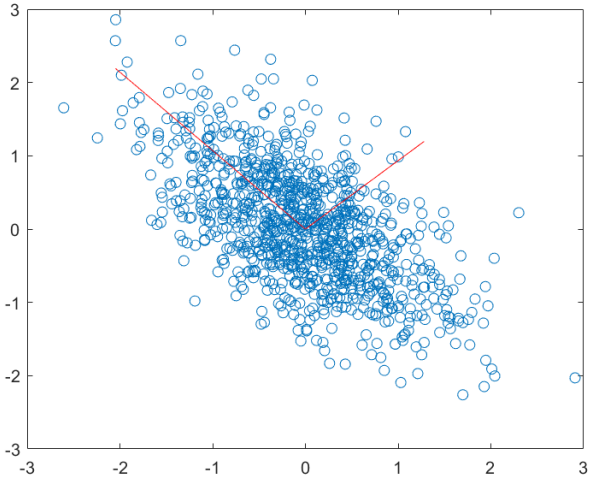

FIGURE 5. Illustration of Gaussian data with differing directions of variance. The red lines are first (up and to the left) and second (up and to the right) principal components from PCA.

a black and white image is taken, each pixel may be assigned a value between 0 (black) and 255 (white). It follows that a single image can be viewed as a vector in $\mathbb{R}^m$ with $m = 12,000,000$. If we have a large collection of images of this size, then we would have a very high-dimensional data set. High-dimensional data can be difficult to handle for many reasons: it could be difficult to store in memory, it could be very costly to compute the SVD of a large data matrix, it could be difficult to ascertain which of the enormous amount of features is important.

Consequently, we often preprocess data by trying to reduce its dimensionality. The general goal of any dimensionality reduction method is to find a "good" embedding $\phi : \mathbb{R}^m \to \mathbb{R}^k$ with $k \ll m$. What is meant by the word good is ambiguous and often dependent on context. Given such a map, we forego A and choose to work with

$$\{\phi(A_{:i})\}_{i=1}^n \subset \mathbb{R}^k,$$

which is a lower-dimensional data set.

Why might this be beneficial? Some reasons include storage savings, computational savings, and to identify important dimensions or features within the data. But one excellent reason that we might consider dimensionality reduction in general is the case when $k = 2$ or 3. After all, we can only visualize things in 2 or 3 dimensions, so if we can find a good map which puts our data into $\mathbb{R}^2$ or $\mathbb{R}^3$, then we can actually see it!

How can we reduce the dimension via PCA? Well, note that $U_k^T : \mathbb{R}^m \to \mathbb{R}^k$, so $U_k^T \widehat{A} \in \mathbb{R}^{k \times n}$. Thus the principal components provide an embedding of $\widehat{A}$ into $\mathbb{R}^k$.

7.2. **Why do we Center the Data?** One natural question one might ask about the PCA algorithm is: why do we need to center the data? The short answer is that if we don't then it could lead to strange behavior in which we are not learning what we hope we are. To see this, note that principal components *always* pass through the origin. But what if our data did not? Then what are we seeing?

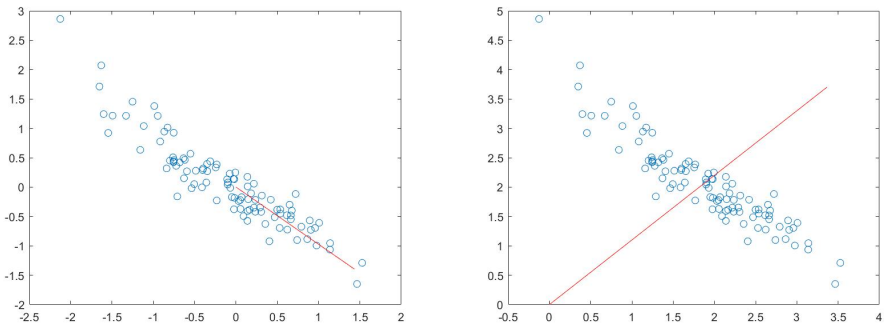

FIGURE 6. PCA with centered data (left) and uncentered data (right). The red lines are the first principal components. Note that in the uncentered case, this no longer represents the direction of largest variance.

8. Lecture 9: Data Clustering Algorithms

8.1. **Visualization.** So you want to learn something about your data? First, you *must* visualize it. Why? Consider Figure 7 below.

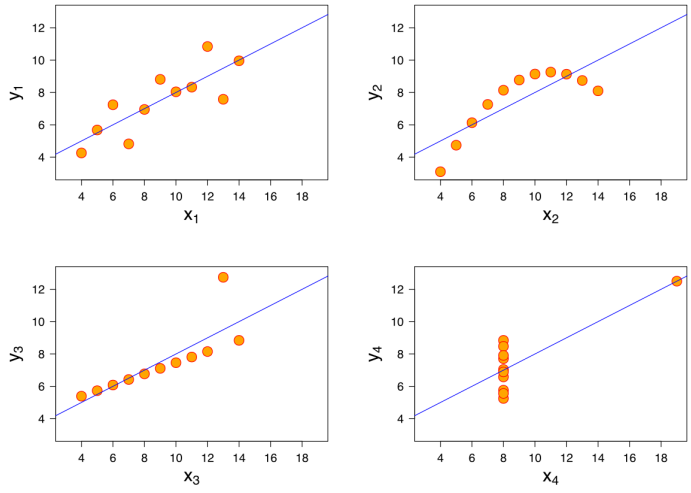

FIGURE 7. Anscombe's quartet. Each data set shown above has the exact same mean in each coordinate direction, variance in each coordinate direction, and linear regression line. (Image source – Wikipedia: https://en.wikipedia.org/wiki/Anscombe%27s_quartet.)

Figure 7 shows four data sets with exactly the same summary statistics, but which exhibit extroardinarily different behaviors, and is due to Anscombe [2]. For a catchier, more modern exhibition of this principle, we refer to the "Datasaurus Dozen" of Matejka, and Fitzmaurice [13] shown in Figure 8.

The point of these figures is a cautionary tale: while statistics are useful in describing data, they are not sufficient, and visualizing our data is of paramount importance. This creates an immediate problem for us given that most data is high-dimensional, and we are limited to viewing things in at most 3 dimensions. Thus, to visualize high-dimensional data sets, one option is to reduce it to a smaller dimension. We mentioned this briefly in Section 7, and will have an ongoing discussion about dimensionality reduction throughout the rest of the course as we develop more tools.

8.2. **Clustering Data.** Another way to understand our data is to try to find *patterns*, or *communities*, within it. For instance, we might want to break plant genomic data into many categories for classification, or split coding languages into a taxonomy based on certain features.

In general, let us consider how to partition our given data $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^m$ into k disjoint partitions $\{A_i\}_{i=1}^k$ such that

$$A_1 \sqcup A_2 \sqcup \cdots \sqcup A_k = X$$

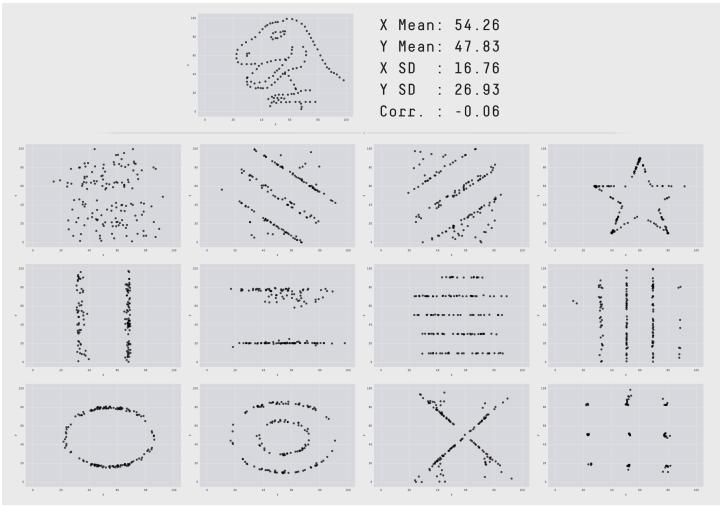

FIGURE 8. The "Datasaurus Dozen" of 13 (including the dinosaur) scatter plots of datasets with the same statistics from [13]; source: https://www.autodeskresearch.com/publications/samestats.

(here $\sqcup$ means disjoint union). While there are feasibly many ways to do this, one way is to try to minimize the sum of the square of distances within each cluster given by:

$$f(A_1,\dots,A_k) := \sum_{j=1}^{k} \sum_{x_i \in A_j} \left| x_i - \frac{1}{|A_j|} \sum_{x_\ell \in A_j} x_\ell \right|^2. \tag{6}$$

Note that the term $|A_j|^{-1} \sum_{x_\ell \in A_j} x_\ell$ in (6) is simply the centroid (center of mass) of the points in A_j, and inside the second sum, we are computing the Euclidean distance from all points in A_j to the center of this cluster.

It is also important to note that f is a function from the set of all possible disjoint partitions of X into k pieces (with no restriction on size) into $\mathbb{R}$. The domain here is discrete, and hence the implicit minimization problem we want to solve ((7) below) is extremely challenging (in fact, it is NP–hard in almost all circumstances).

$$\min_{A_1 \sqcup \cdots \sqcup A_k = X} f(A_1,\dots,A_k). \tag{7}$$

The function f is called the k–means **objective function**, and it should be noted that we also have a statistical interpretation of f as follows:

$$f(A_1,\dots,A_k) = \sum_{j=1}^{k} |A_j| \mathrm{Var}(A_j).$$

So minimizing the k–means objective function boils down to minimizing the sum of the product of the size of A_j and the variance (in the usual statistical sense) of the data points within A_j. We leave it as an exercise to the reader to verify this reformulation of f.

8.3. ***k*–means and Lloyd's Algorithm.** What we have not discussed yet is how we can approximate the solution to (7). There are various ways to do this, but the classical one we posit here is called Lloyd's Algorithm after Stuart P. Lloyd [12]. As an interesting historical note, Lloyd published his algorithm internally at Bell Labs in 1957, but it did not appear in the rest of the literature until Forgy rediscovered it in 1965 a whole 8 years later [9]. Lloyd was not credited with invention of the algorithm until much later when his paper was released to the public and published by IEEE in 1982.

The idea of Lloyd's Algorithm is based on simple geometry: if we are given a partition of the data $A_1, \ldots, A_k$, we can compute *Voronoi regions* based on the centroids of the data appearing within each partition. That is, if $A_i \subset X$ and μ_i is its center ($\mu_i = |A_i|^{-1} \sum_{x_j \in A_i} x_j$), then its associated Voronoi region is the set

$$V_i := \left\{ x \in \mathbb{R}^m : |x - \mu_i| \leq |x - \mu_j|, \quad j \neq i \right\}. \tag{8}$$

Note that $\{V_i\}_{i=1}^k$ partition the whole ambient space, namely $\bigcup_{i=1}^k V_i = \mathbb{R}^m$. This partition is not disjoint, as the regions can intersect on hyperplanes of dimension $m-1$. For illustration, Figure 9 shows a set of data which has already been clustered, and illustrates the associated Voronoi regions.

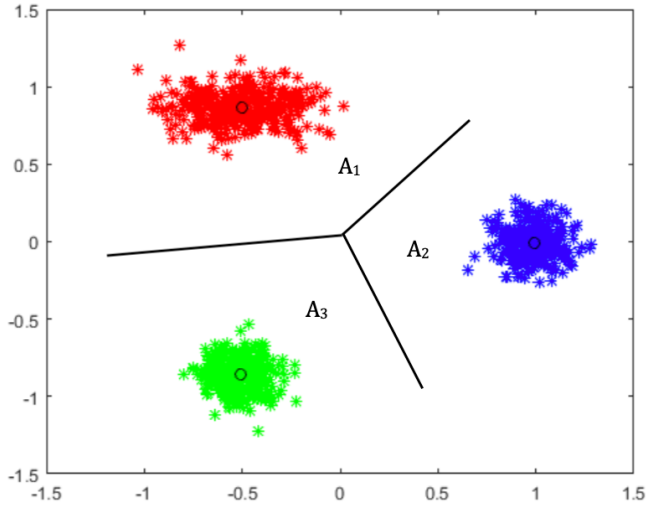

FIGURE 9. Example of Voronoi regions (boundaries shown by straight lines) providing geometric clustering for the case $k = 3$ with data in $\mathbb{R}^2$.

Clearly Figure 9 achieved the correct clustering for this simple data set that matches how any human would partition the data. The reader should consider that for this partitioning, the function $f(A_1, A_2, A_3)$ should indeed be small. Figure 10 shows a simple data set which has been partitioned two different ways. It should be evident that the objective function f should have a much smaller value for the left partitioning than it would for the right.

The question then is: how can we find these Voronoi regions? Here is the simplest version of Lloyd's Algorithm which tries to achieve this.

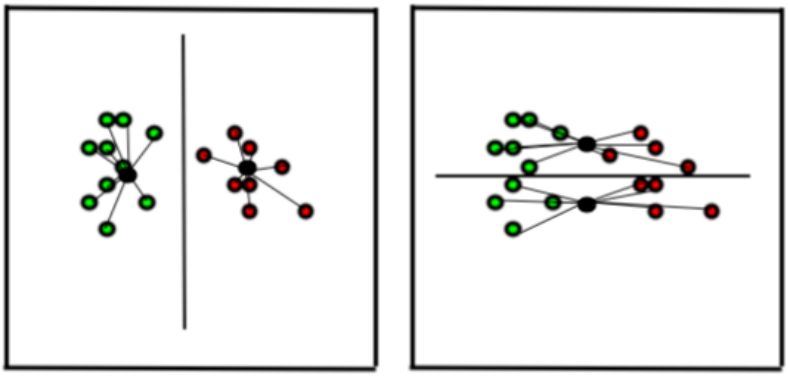

FIGURE 10. Data set with the "correct" partitioning corresponding to green and red color. Good division via Voronoi regions (left) leading to small k–means objective function and poor division (right) leading to a larger objective function.

Algorithm 2: Lloyd's k–means Approximation Algorithm

Input : Data $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^m$, number of clusters $k \in \mathbb{N}$, tolerance TOL, and maximum number of iterations $N_{\max}$
Output: Partition $A_1, \ldots, A_k$ such that $A_1 \sqcup \cdots \sqcup A_k = X$
Initialize centers $\mu_1, \ldots, \mu_k$
Compute Voronoi regions $V_1, \ldots, V_k$ according to centers $\{\mu_i\}$ (Equation (8))
$A_i = X \cap V_i, \quad i = 1, \ldots, k$
while iter $< N_{\max}$ **do**
  $\mu_i' = \frac{1}{|A_i|} \sum_{x_\ell \in A_i} x_\ell$
  Update Voronoi regions $V_1, \ldots, V_k$ according to centers $\{\mu_i'\}$
  $A_i = X \cap V_i$
  if $|\mu' - \mu| <$ TOL **then**
    break
  else
    $\mu_i = \mu_i'$
    iter = iter + 1
  end
end
return $A_1, \ldots, A_k$

8.4. **A Note on Initialization.** The Initialization step for Lloyd's Algorithm deserves some discussion here. One way to do this is to simply randomly select the centers from the uniform distribution on the smallest cube in $\mathbb{R}^n$ that contains the data X. However, it has been known by practitioners for quite some time that this is typically a poor choice of initialization in almost all circumstances, and it's easy to see why given that this method fails to take into account any feature of the data.

Another common method is the so-called k–means++ initialization [3] (which is Matlab's standard choice). This method begins by choosing μ_1 uniformly at random from X (i.e., chooses $x_i \in X$ with probability $\frac{1}{n}$). Next, μ_2 is chosen from $X \setminus \{x_i\}$, where

$\mu_2 = x_j \in X \setminus \{x_i\}$ with probability proportional to $|x_j - \mu_1|^2$. We iteratively randomly choose $\mu_3, \ldots, \mu_k$ in a similar fashion where at each stage we compute the distance of a remaining unchosen data point to the nearest center and create a probability distribution from this. This initialization typically gives better behavior than uniform random initialization due to the fact that we are taking into account some structural features of the data when selecting initial centers.

How important is initialization, you ask? Well, a good initialization can lead not only to good clustering, but also to fewer iterations in the algorithm, whereas a poor initialization can lead to poor clustering performance or a large number of iterations before convergence. Figure 11 gives an example of this fact.

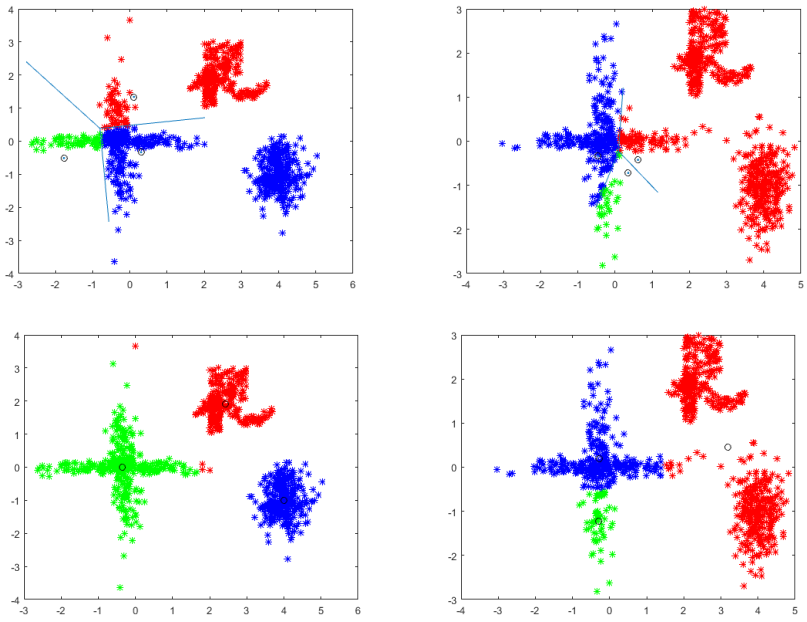

FIGURE 11. Data with three clusters and different center initializations (top). Final output of Lloyd's Algorithm (Algorithm 2) given these initial centers (bottom). The first initialization gives a mostly accurate final clustering (bottom left) whereas the second yields poor clustering by putting two distinct clusters together (bottom right). Dataset generated with code from Alex Powell.

9. Graphs From Data

Let's pause for a moment and consider a basic fact about the k–means algorithm and objective function that was discussed in the previous section. By choosing that objective function to minimize, we made an implicit assumption about the clusters appearing within our data:

the notion of "similarity" within the data is *pairwise distance*.

The k–means objective function (6) attempts to minimize the within-cluster distances, but we must ask the question: is this always the correct notion of similarity? Consider the following example in Figure 12.

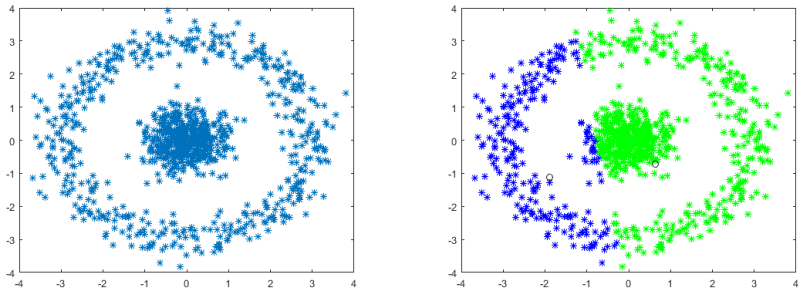

FIGURE 12. A data set consisting of points from a disk and an annulus.

Human beings will easily be able to tell you that there are two distinguishable clusters in the data set illustrated in Figure 12: the disk, and the surrounding annulus. However, if we try to run Lloyd's Algorithm on this data with $k = 2$, we will obtain something like the right-hand side. So what happened? Again, our notion of similarity within the data was pairwise distance, but that is not the correct notion of similarity for this data. Indeed, a point on the top of the annulus is much farther from a point on the bottom of the annulus than it is from a point in the center of the disk. Another way to put this is that this data is not *linearly separable* in two dimensions, meaning we cannot draw a line between the two clusters.

The point here is: in a clustering task, choosing the right notion of similarity is data-dependent, and typically not simple to do beforehand. We will turn our discussion to Spectral Clustering, which is a method that 1) can separate the disk and annulus clusters easily, and 2) takes into account the structure of the data in a more general way than k–means does. But to get to the algorithm, we need to make a detour through Graph Theory, and discuss how to make graphs from data sets.

9.1. Graphs.

Much of the material here and on Spectral Clustering here is taken from the excellent tutorial [17].

Definition 9.1. *A **graph** $G = (V, E)$ is a collection of vertices $V = \{v_1, \ldots, v_n\}$, some of which are connected by an edge in the set $E \subset V \times V$.*

*A graph is **weighted** if there is a nonnegative weight function $w : E \to \mathbb{R}$ which is not identically 1.*

*A graph is **unweighted** if all edges are assigned weight 1.*

*A graph is **undirected** if edges have no direction, i.e., $w(v_i, v_j) = w(v_j, v_i)$.*

For our purposes, we will exclusively consider undirected graphs which may or may not be weighted depending on the situation.

9.2. **Graphs From Data.** Suppose we are given data $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^m$. How can we make a graph based upon this data? First, we simply take $V = \{x_1, \dots, x_n\}$, or in other words, each data point is a vertex in the graph. Now we must determine a way to assign edges between vertices. There are several standard ways to do this.

Euclidean Graph. The Euclidean (or geometric) graph over the vertices is the one in which we connect all points to every other point, and assign weights via

$$w_{ij} := w(x_i, x_j) = |x_i - x_j|.$$

This results in a fully connected graph with small weights for nearby vertices and large weights for distant ones.

ε–neighborhood graph. Given a parameter $\varepsilon > 0$, we put a ball of radius ε around each vertex, and connect the vertex to every other vertex that lies in this ball. Thus the weight function is

$$w_{ij}^{\varepsilon} := \begin{cases} 1, & |x_i - x_j| \leq \varepsilon \\ 0, & \text{otherwise.} \end{cases}$$

k–nearest neighbor (k-NN) graph. Given a fixed $k \in \mathbb{N}$, for $i = 1, \dots, n$, we define K_i to be the set of k nearest neighbors of the point x_i (i.e., the k points in X which are closest in distance to x_i). Then we set

$$w_{ij}^{k\text{NN}} := \begin{cases} 1, & x_j \in K_i \text{ or } x_i \in K_j \\ 0, & \text{otherwise.} \end{cases}$$

Mutual k–NN. Note that if you are one of my two nearest neighbors, it doesn't follow that I am one of your two nearest neighbors (this is why we had the "or" statement in the weight definition for k–NN above). But, we can form a more restrictive graph by making edges between points only when they are both k–nearest neighbors of each other. To wit, we have

$$w_{ij}^{\text{M}k\text{NN}} := \begin{cases} 1, & x_j \in K_i \text{ and } x_i \in K_j \\ 0, & \text{otherwise.} \end{cases}$$

Gaussian weighted graph. Given a parameter $\sigma > 0$, the Gaussian weighted graph assigns weights as

$$w_{ij}^{\sigma} := e^{-\frac{|x_i - x_j|^2}{\sigma^2}}.$$

Thus, a Gaussian weighted graph has strong connections for very close vertices, and rapidly weaker connections as vertices become farther apart. Each of these graphs is used heavily in many applications (possibly with some modifications such as weighting edges in neighbor graphs). We will discuss their use in clustering in the next lecture.

9.3. **Attributes of Graphs.**

Definition 9.2. *Given a weighted graph $G = (V, E, w)$, the **degree** of a vertex $v_i \in V$, is defined to be*

$$d_i := \sum_{j=1}^{n} w_{ij}.$$

*The **degree matrix** is the matrix $D := \text{diag}(d_1, \dots, d_n)$. The **weight matrix** or **adjacency matrix** is given by $W_i j = w(v_i, v_j)$, where w is the weight function assigning edge weights.*

In the above definition, the degree of a vertex is the sum of the weights of edges which connect to the vertex. So for an unweighted graph, this just counts the number of edges incident to the given vertex.

Note that in an undirected graph, the matrix W is symmetric.

Definition 9.3. *Given a weighted graph $G = (V, E, w)$, the **unnormalized graph Laplacian** is the matrix $L = D - W$.*

Example 9.4. Consider a graph with a vertex in the middle, labeled v_1, that is connected to 4 vertices surrounding it (so the corners of a square connected to the center points of the square). The corner vertices will be labeled $v_2, \dots, v_5$, and are only connected to v_1. If we assume all weights are 1, then we have that $D = \operatorname{diag}(4,1,1,1,1)$, and

$$W = \begin{bmatrix} 0 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Hence the unnormalized graph Laplacian in this case is

$$L = D - W = \begin{bmatrix} 4 & -1 & -1 & -1 & -1 \\ -1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Something important to not here: L is symmetric and diagonally dominant, hence it is positive semi-definite. Let us make some additional important notes on the properties of the graph Laplacian.

Theorem 9.5. *Given a weighted graph $G = (V, E, w)$ with L being its unnormalized graph Laplacian, the following hold.*

(1) For all $x \in \mathbb{R}^n$, $\langle Lx, x\rangle = \frac{1}{2}\sum_{i,j=1}^n w_{ij}(x_i - x_j)^2$,

(2) L is symmetric, positive semi-definite,

(3) The smallest eigenvalue of L is 0 with corresponding eigenvector $\mathbb{1} := \begin{bmatrix} 1 & \cdots & 1 \end{bmatrix}^T$,

(4) L has non-negative, real eigenvalues $0 = \lambda_n \le \cdots \le \lambda_1$.

Proof. (1): Note that $\langle Lx, x\rangle = \langle (D-W)x, x\rangle = \langle Dx, x\rangle - \langle Wx, x\rangle$, and that since D is diagonal, we have that $Dx = \begin{bmatrix} d_1 x_1 & \dots & d_n x_n \end{bmatrix}^T$. Thus

$$\begin{aligned} \langle Lx, x\rangle &= \sum_{i=1}^n d_i x_i^2 - \sum_{i,j=1}^n x_i x_j w_{ij} \\ &= \frac{1}{2}\left(\sum_{i=1}^n d_i x_i^2 - 2\sum_{i,j=1}^n x_i x_j w_{ij} + \sum_{j=1}^n d_j x_j^2\right) \\ &= \frac{1}{2}\left(\sum_{i,j} = 1^n w_{ij} x_i^2 - 2\sum_{i,j=1}^n x_i x_j w_{ij} + \sum_{i,j=1}^n w_{ji} x_j^2\right) \end{aligned}$$

where the last line followd from the fact that $d_i = \sum_{j=1}^n w_{ij}$. Now combining terms, we have

$$\langle Lx, x\rangle = \frac{1}{2}\sum_{i,j=1}^n w_{ij}(x_i - x_j)^2$$

which is the desired conclusion.

(2): Note that $L^T = D^T - W^T = D - W$, and hence is symmetric, while combining the fact that $w_{ij} \geq 0$ with (1) implies that $\langle Lx, x\rangle \geq 0$ for all $x \in \mathbb{R}^n <$ and so L is symmetric positive semi-definite.

(4) follows directly from (2), and to show (3), notice that

$$(D-W)\begin{bmatrix}1\\ \vdots \\ 1\end{bmatrix} = \begin{bmatrix} d_1 - \sum_{i=1}^n w_{1i} \\ \vdots \\ d_n - \sum_{i=1}^n w_{ni}\end{bmatrix} = 0$$

by definition of d_i. □

There are multiple options for defining graph Laplacians that come from different starting points. We will define two here, but focus on one of them due to its utility for clustering.

Definition 9.6. *Given a weighted graph $G = (V, E, w)$, the **symmetric normalized graph Laplacian** is defined by*

$$L_{\text{sym}} := I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}} = D^{-\frac{1}{2}} L D^{-\frac{1}{2}}.$$

*The **random walk graph Laplacian** is defined by*

$$L_{\text{rw}} := I - D^{-1} W = D^{-1} L.$$

The salient properties of these versions are as follows.

Theorem 9.7. *Given a weighted graph $G = (V, E, w)$, the following hold.*

(1) For all $x \in \mathbb{R}^n$, $\langle L_{\text{sym}} x, x\rangle = \frac{1}{2}\sum_{i,j=1}^n w_{ij}(\frac{x_i}{\sqrt{d_i}} - \frac{x_j}{\sqrt{d_j}})^2$,

(2) (λ, u) is an eigenvalue/eigenvector pair of L_{rw} if and only if $(\lambda, D^{\frac{1}{2}}u)$ is an eigenvalue/eigenvector pair of L_{sym},

(3) $(0, \mathbb{1})$ is an eigenvalue/eigenvector pair of L_{rw}, and $(0, D^{\frac{1}{2}}\mathbb{1})$ is an eigenvalue/eigenvector pair of L_{sym},

(4) L_{sym} and L_{rw} are symmetric, positive semi-definite and have non-negative, real eigenvalues $0 = \lambda_n \leq \cdots \leq \lambda_1$.

The proof follows much the same lines as that of Theorem 9.5, and so we leave it as an exercise to the motivated reader.

Let us first state a proto-algorithm which encompasses the essential variants of Spectral Clustering.

Algorithm 3: Spectral Clustering

Input : Data $X = \{x_i\}_{i=1}^n \subset \mathbb{R}^m$, number of clusters $k \in \mathbb{N}$
Output: Partition $A_1, \dots, A_k$ such that $A_1 \sqcup \cdots \sqcup A_k = X$
Form a graph $G = (V, E, w)$ from X (see Section 9.2)
Let $\widetilde{L}$ be any graph Laplacian of G
Compute eigenvectors of $\widetilde{L}$ corresponding to the smallest k eigenvalues (call these $u_n, \dots, u_{n-k+1}$)
Let $A_1, \dots, A_k$ be the output of Lloyd's Algorithm (Algorithm 2) on the matrix $U = \begin{bmatrix} u_n & \cdots & u_{n-k+1} \end{bmatrix}^T$
return $A_1, \dots, A_k$

We will discuss briefly later the effect of what graph and associated graph Laplacian one chooses has on the clustering outcomes, but for now, we note that this algorithm at present seems unmotivated, which can be overcome by a brief interlude on graph cut problems. For now, let us note that Algorithm 3 can separate the disk and annulus we showed previously.

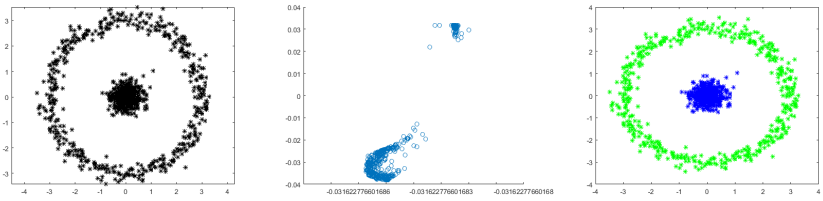

FIGURE 13. Disk and Annulus data set (left) with the columns of $U \in \mathbb{R}^{2\times n}$ as in Algorithm 3 (middle) where an ε–neighborhood graph was used along with the symmetric normalized graph Laplacian, and the clustering output color-coded (right). We see that Spectral Clustering with these parameters obtained 100% accuracy in this particular run.

10.1. **Graph Cuts.** Graph Cut problems are not necessarily related to clustering problems *a priori*, but Spectral Clustering turns out to be the marriage of the two ideas. In general, given a graph $G = (V, E)$, the graph cut problem is to find the "best" partition of a graph into 2 or more disjoint pieces. Here, the notion of a good partition is up for grabs, but we will give several candidates.

Definition 10.1. *Given a (possibly weighted) graph $G = (V, E, w)$, and a nonempty subset of the vertices $A \subset V$, the **Cut quantity** of A is defined to be*

$$\mathrm{Cut}(A, A^C) := \sum_{v_i \in A, v_j \in A^C} w(v_i, v_j).$$

The cut quantity is a measure of the weight of edges crossing the boundary of the partition, so the smaller the cut quantity, the lower total weight is being cut by the partition boundary, which hopefully means that we are not cutting through very dense parts of the graph. With this in mind, we can formulate the following problem.

Problem 1 (MinCut Problem)**.** Given a (possibly weighted) graph $G = (V, E, w)$, solve

$$\operatorname*{argmin}_{A \subset V} \operatorname{Cut}(A, A^C).$$

There is one main issue with the MinCut Problem as it is stated here, and that is that for many graphs, the optimal solution to this problem takes $A = \{v\}$ and isolates a single vertex. The reason for this is that we made no prescription in our definition of the Cut quantity for how large a subset A of the vertices must be. In an effort to counteract this bad behavior, we can define a couple of different cut quantities as follows.

Definition 10.2. *Given a (possibly weighted) graph $G = (V, E, w)$, and a nonempty subset of the vertices $A \subset V$, the **RatioCut quantity** of A is defined to be*

$$\operatorname{RatioCut}(A, A^C) = \frac{\operatorname{Cut}(A, A^C)}{|A|} + \frac{\operatorname{Cut}(A, A^C)}{|A^C|},$$

where $|A|$ is the cardinality of A even if G is weighted.

Problem 2 (RatioCut Problem)**.** Given a (possibly weighted) graph $G = (V, E, w)$, solve

$$\operatorname*{argmin}_{A \subset V} \operatorname{RatioCut}(A, A^C).$$

The RatioCut Problem tries to achieve a balanced partition of the graph, where balance here means that we want the weight of crossings to be small, but also we prioritize partitions where $|A|$ and $|A^C|$ are roughly similar. We note that for unweighted graphs, this is completely natural, but $|A|$ completely fails to take into account the weights of the edges, whereas $\operatorname{Cut}(A, A^C)$ does. So for weighted graphs it is natural to consider the following quantity.

Definition 10.3. *Given a (possibly weighted) graph $G = (V, E, w)$, and a nonempty subset of the vertices $A \subset V$, the **Normalized Cut (NCut) quantity** of A is defined to be*

$$\operatorname{NCut}(A, A^C) = \frac{\operatorname{Cut}(A, A^C)}{\operatorname{Vol}(A)} + \frac{\operatorname{Cut}(A, A^C)}{\operatorname{Vol}(A^C)},$$

where $\operatorname{Vol}(A) := \sum_{i \in A} d_i$.

Problem 3 (Normalized Cut (NCut) Problem)**.** Given a (possibly weighted) graph $G = (V, E, w)$, solve

$$\operatorname*{argmin}_{A \subset V} \operatorname{NCut}(A, A^C).$$

The NCut problem again seeks balanced clusters, but this time balance means we want the volumes of A and its complement to be similar, where the volume of a set of vertices is the total weight of edges incident to it (recall the degree of a vertex in a weighted graph is the sum of weights of edges attached to it). Note that even for unweighted graphs, the volume is not the same as the cardinality, and so these quantities are different in general.

10.2. **Spectral Clustering as a Relaxation of Graph Cuts.** Now let's see one case in which we can recover the Spectral Clustering algorithm as a relaxation of a graph cut problem. Specifically, we will focus on how Unnormalized Spectral Clustering (i.e., using L, the unnormalized graph Laplacian) is a convex relaxation of the RatioCut Problem. For exposition, we will consider the case of $k = 2$ clusters.

Recall that RatioCut seeks to solve the following:

$$\min_{A \subset V} \text{RatioCut}(A, A^C) = \min_{A \subset V} \frac{\text{Cut}(A, A^C)}{|A|} + \frac{\text{Cut}(A, A^C)}{|A^C|}. \tag{9}$$

In an effort to rewrite this in terms of the graph Laplacian, suppose we have a set $A \subset V$, and let $x \in \mathbb{R}^n$ have entries

$$x_i = \begin{cases} \sqrt{\dfrac{|A^C|}{|A|}}, & i \in A \\ -\sqrt{\dfrac{|A|}{|A^C|}}, & i \in A^C. \end{cases} \tag{10}$$

By Theorem 9.5(1), we have

$$\begin{aligned} \langle Lx, x \rangle &= \frac{1}{2} \sum_{i,j=1}^{n} w_{ij}(x_i - x_j)^2 \\ &= \frac{1}{2} \sum_{i \in A, j \in A^C} w_{ij} \left(\sqrt{\frac{|A^C|}{|A|}} + \sqrt{\frac{|A|}{|A^C|}} \right)^2 + \frac{1}{2} \sum_{i \in A^C, j \in A} w_{ij} \left(-\sqrt{\frac{|A|}{|A^C|}} - \sqrt{\frac{|A^C|}{|A|}} \right)^2. \end{aligned}$$

Note that we do not have any contribution from terms in which $i, j \in A$ or $i, j \in A^C$ since in these cases $x_i = x_j = 0$. Now pulling out constants, we have

$$\langle Lx, x \rangle = \frac{1}{2} \left(\sqrt{\frac{|A^C|}{|A|}} + \sqrt{\frac{|A|}{|A^C|}} \right)^2 \sum_{i \in A, j \in A^C} w_{ij} + \frac{1}{2} \left(\sqrt{\frac{|A|}{|A^C|}} + \sqrt{\frac{|A^C|}{|A|}} \right)^2 \sum_{i \in A^C, j \in A} w_{ij}.$$

But notice that both summations are $\text{Cut}(A, A^C)$ as this quantity is symmetric in its definition. Thus this term can be factored, and noting that the squared terms are also identical, a bit of algebra reveals

$$\begin{aligned} \langle Lx, x \rangle &= 2 \frac{1}{2} \left(\frac{|A^C|}{|A|} + \frac{|A|}{|A^C|} + 2 \sqrt{\frac{|A^C||A|}{|A||A^C|}} \right) \text{Cut}(A, A^C) \\ &= \left(\frac{|A^C|}{|A|} + \frac{|A|}{|A^C|} + 2 \right) \text{Cut}(A, A^C) \\ &= \left(\frac{|A^C|}{|A|} + \frac{|A|}{|A^C|} + \frac{|A|}{|A|} + \frac{|A^C|}{|A^C|} \right) \text{Cut}(A, A^C) \\ &= \left(\frac{|A^C| + |A|}{|A|} + \frac{|A| + |A^C|}{|A^C|} \right) \text{Cut}(A, A^C) \\ &= \left(\frac{n}{|A|} + \frac{n}{|A^C|} \right) \text{Cut}(A, A^C) \\ &= n \left(\frac{\text{Cut}(A, A^C)}{|A|} + \frac{\text{Cut}(A, A^C)}{|A^C|} \right) \\ &= n \text{RatioCut}(A, A^C). \end{aligned}$$

The third line is simply rewriting 1 in various forms to add to the existing terms, whereas the third to last line follows from the fact that $V = A \cup A^C$, hence $n = |V| = |A| + |A^C|$; the final equality is by definition.

Let us also note that for x of the form (10), we have

$$\langle x, \mathbb{1} \rangle = \sum_{i=1}^{n} x_i = \sum_{i \in A} \sqrt{\frac{|A^C|}{|A|}} - \sum_{i \in A^C} \sqrt{\frac{|A|}{|A^C|}} = |A| \sqrt{\frac{|A^C|}{|A|}} - |A^C| \sqrt{\frac{|A|}{|A^C|}} = 0,$$

so x is orthogonal to the all ones vector $\mathbb{1}$.

Additionally, we have

$$|x|^2 = \sum_{i=1}^{n} x_i^2 = |A| \frac{|A^C|}{|A|} + |A^C| \frac{|A|}{|A^C|} = |A^C| + |A| = n.$$

Combining all these observations together, we have that (9) is equivalent to

$$\min_{A \subset V} \frac{\langle Lx, x \rangle}{\langle x, x \rangle}, \quad \text{subject to } x \text{ has the form (10)}, \ \langle x, \mathbb{1} \rangle = 0, |x| = \sqrt{n}. \tag{11}$$

The problem with trying to solve (11) directly is that it is minimizing over a discrete set of all possible partitions of V into two disjoint subsets, and this turns out to be NP–hard to solve exactly. A common trick to try to get a solution here is to form what is called the *convex relaxation* of the problem (11). The relaxation in this case is

$$\min_{x \in \mathbb{R}^n} \frac{\langle Lx, x \rangle}{\langle x, x \rangle}, \quad \text{subject to } \langle x, \mathbb{1} \rangle = 0, \ |x| = \sqrt{n}. \tag{12}$$

This is a Rayleigh quotient, which we have seen in a previous homework assignment, and we know the solution is $x = u_{n-1}$, the $(n-1)$–st eigenvector of L corresponding to the eigenvalue λ_{n-1} (this is because $(0, \mathbb{1})$ is the smallest eigenvalue/eigenvector pair of L by Theorem 9.5. This justifies the use of $\begin{bmatrix} u_n & u_{n-1} \end{bmatrix}$ in Algorithm 3.

So in conclusion, the SVD of the unnormalized graph Laplacian solves (12), and in this sense, Unnormalized Spectral Clustering is a convex relaxation of the RatioCut Problem.

One natural question that arises is: given a solution to the relaxed problem (12), how do we infer a partition A? One simple way to do this is to take

$$i \in \begin{cases} A, & x_i \geq 0 \\ A^C, & x_i < 0. \end{cases}$$

For $k > 2$, a similar argument will show that the eigenvectors $u_{n-1}, u_{n-2}, \ldots, u_{n-k+1}$ give the solution to the Relaxed optimization problem (note that a slightly different version of Rayleigh quotients needs to be used in this case; see [17] for a thorough derivation).

Also note that if we use L_{sym} instead of L, then this is sometimes called Normalized Spectral Clustering, and the equivalent formulation of (12) shows that this is a relaxation of the NCut problem.

10.3. **How do the solutions compare?** A strong word of caution is in order here: solutions to (11) and (9) need not be the same, nor even similar.

Figure 14 illustrates this phenomenon on so-called "Cockroach Graphs."

It requires a proof that the solutions are as stated in Figure 14; for the precise references see [17]. This example shows that in some cases, the value of the objective function RatioCut(A, A^C) for the relaxed solution can be as far apart as possible from that of the real solution, as the values differ by a factor of $\frac{1}{2}k = O(n)$ in the example.

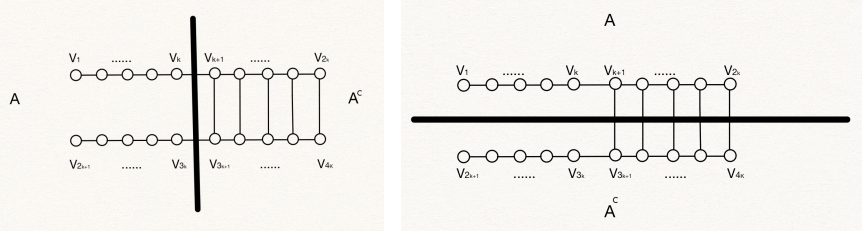

FIGURE 14. (Left) The solution to (11) with $\text{RatioCut}(A, A^C) = \frac{2}{2k} + \frac{2}{2k} = \frac{2}{k}$. (Right) The solution to (12) with $\text{RatioCut}(A, A^C) = \frac{k}{2k} + \frac{k}{2k} = 1$.

10.4. **Algorithmic Aspects.** Here, we give a very brief discussion of some of the algorithmic choices that need to be made in running Spectral Clustering. For a more thorough discussion, see [17, Section 8]. There are three main issues that one must consider:

(1) How do we estimate k (the number of clusters)?
(2) Which graph should we form for the data?
(3) Which graph Laplacian should be used?
(4) How do we compute the small eigenvalues and eigenvectors of the graph Laplacian?

We will discuss these questions in the order posed here.

10.4.1. *Estimating k.* First, let us return to our discussion k–means, and note that there we may use something known as the *elbow method* to estimate k as follows.
Step 1: Run Lloyd's Algorithm for a range of k values (from 1 to some predetermined maximum).
Step 2: Plot the objective function (6) against k and find the "elbow".

This is imprecise in general, but Figure 15 illustrates what one is hoping for in this method.

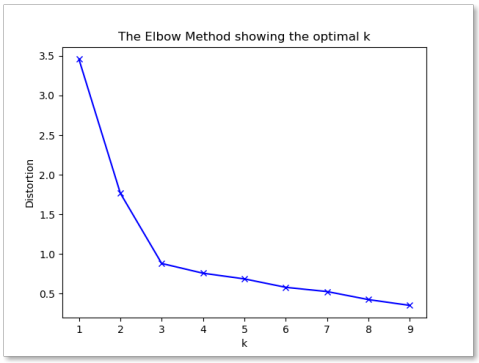

FIGURE 15. k–means objective function vs. k. Note that one hopes for a significant bend (i.e., sharp change in slope) such as is seen at $k = 3$ here.

Of course this example is not always true to reality, and there are many instances where one would have a steady decrease in the objective function, and there would be

no discernible bend, or elbow. In that case, the clustering task is, in some sense, not well-posed.

Now for Spectral Clustering, there is another feasible way to estimate the number of clusters. Suppose that we have chosen to use either the ε–neighborhood of k–NN graph for the data. Then if we did well, we may have k distinct connected components in our graph corresponding to k distinct clusters. In that case, we may use the following result to discern what k is from the graph Laplacian.

Theorem 10.4. *Let G be an undirected, weighted graph. If its graph Laplacian, L, has an eigenvalue of 0 with multiplicity k, then G has k connected components.*

Proof. See [17, Proposition 2]. □

Theorem 10.4 tells us that we need only find the multiplicity of the 0 eigenvalue of L (note that L_{rw} and L_{sym} work as well) and this will tell us the number of clusters.

10.4.2. *Which graph should be used?* This is somewhat dependent upon the data unfortunately, but k–NN graphs tend to give the most faithful information about the data. On the other hand, in many applications, practitioners tend toward using the Gaussian-weighted fully connected graph. In any case, there is no formulaic or theoretical way to choose the best graph to use for a given data set, unfortunately.

10.4.3. *So what about the graph Laplacian?* The real question here is: which graph Laplacian leads to meaningful eigenvalues? Again, there is no unanimous consent here, but Bertozzi and Flenner [6] make a convincing argument that the symmetric normalized graph Laplacian, L_{sym} ought to be used. They show that on an image of two cows in a field, the second, third, and fourth eigenvectors of L_{sym} capture information about the background, and the background and foreground cows, respectively, while the eigenvectors of L are apparently meaningless.

10.4.4. *Computing eigenvalues and eigenvectors.* For now, we will shelve the discussion of how to numerically compute the eigenvalues and eigenvectors required in Spectral Clustering until later in favor of jumping to an application of the things we have discussed here.

11. Application: Subspace Clustering

We mentioned before that the implicit assumption in the k–means objective function was that two data points are "similar" if and only if they are close together in the Euclidean sense. However, sometimes data exhibits an altogether different structure which fundamentally alters the notion of similarity in the clustering task. The Subspace Clustering Problem is exactly such a situation. First, let's discuss the model for our data which will be motivated by a particular application shortly.

Definition 11.1 (Union of Subspaces)**.** *A data matrix $X \subset \mathbb{R}^n$ comes from a **union of subspaces** model provided the columns of X lie in $\mathscr{U} := \bigcup_{i=1}^{L} S_i$ where each S_i is a linear subspace of $\mathbb{R}^n$.*

Figure 16 illustrates the model under consideration.

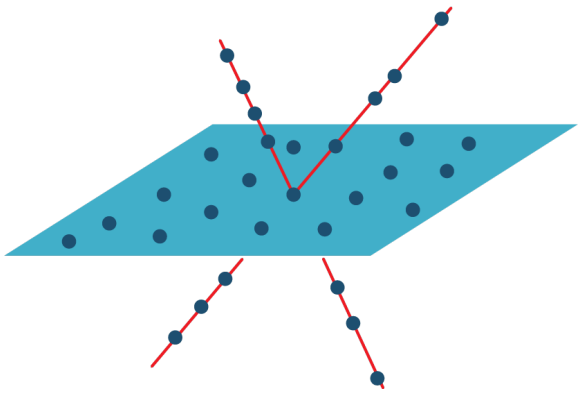

FIGURE 16. An example of the union of subspaces model. Here there are 3 subspaces of dimensions 1, 1, and 2.

Problem 4. The **Subspace Clustering Problem** is: given data $X \in \mathbb{R}^{m\times n}$ whose columns come from a union of subspaces $\mathscr{U}$, find

(1) the number of subspaces, L,
(2) the dimensions of each subspace
(3) a basis for each subspace,
(4) a clustering of the data.

Here, clustering means we want to find an assignment function $\Pi : X \to \{1,\dots,L\}$ which satisfies $\Pi(x_i) = \Pi(x_j) = k$ if and only if x_i and x_j are in the subspace S_k.

It should be noted that finding the clustering of subspace data will determine the rest of the information in the Subspace Clustering Problem; indeed, a basis for each subspace can be obtained by running the Gram–Schmidt Algorithm on the data within each cluster.

11.1. Applications of Subspace Clustering.

Before describing solution methods for the Subspace Clustering Problem, let us give some sample applications in which the data satisfies the union of subspaces model, and the desired task is to cluster the data according to subspaces.

11.1.1. *Motion Segmentation.* Motion segmentation is a task in the area of *Computer Vision*. Suppose we have a camera which is filming a scene; each frame of the video sequence is a still image of a fixed size based on the resolution of the camera. Suppose we have a mechanism of feature selection, in which we select feature points on moving objects in the first frame of the video, and track their positions over the rest of the frames.

More concretely, the camera will have a 3-dimensional coordinate system at each frame, whose axes we will denote by i_f, j_f, k_f (f for the particular frame index). An object we are tracking will also have an object coordinate system, whose axes we will simply denote by x_f, y_f, z_f. Suppose also that t_f is the vector originating at the camera origin and terminating at the object origin. Now, given a point p_f on the object that is represented in object coordinates, we may represent it in camera coordinates via

$$p_f^C = R_f p_f + t_f$$

where

$$R_f = \begin{bmatrix} i_f^T \\ j_f^T \\ k_f^T \end{bmatrix}, \tag{13}$$

and t_f is the translation vector from one coordinate system to the other as described above. The matrix R_f describes the rotation, while t_f describes the translation requires to map the object coordinate axes onto the camera ones.

Now, we will utilize a trick that is abundantly common in computer vision, which comes from some ideas in projective geometry, which is to write all of our points in *homogeneous coordinates*. The point in homogeneous object coordinates is

$$s_f := \begin{bmatrix} p_f \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}. \tag{14}$$

One use of homogeneous coordinates is that any points on a given line through the origin have the same homogeneous coordinates, but another pertinent use for us is that these coordinates allow us to write rotation plus translation as a single matrix multiplication. Indeed, in homogeneous camera coordinates, our point is

$$s_f^C = \begin{bmatrix} R_f & t_f \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p_f \\ 1 \end{bmatrix} = \begin{bmatrix} x_f^C \\ y_f^C \\ z_f^C \\ 1 \end{bmatrix}.$$

Now for a real camera, points along a line passing through the camera origin end up in the same pixel of a still image – this is what leads to perspective in the images. However, we will use a simplified camera model called the *orthographic projection model*, in which we assume that points in the camera coordinate are projected orthogonally onto the $i_f j_f$–plane of the camera (for ease of understanding and notation, let's just call this the $x^C y^C$–plane from here on out). This simplifies our representation a bit, and is generally reasonable for objects filmed at a medium to far distance.

With this camera model, only the x^C, y^C coordinates of a a point in homogeneous camera coordinates end up being captured by the camera, and we obtain

$$\begin{bmatrix} x_f^C \\ y_f^C \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} R_f & t_f \\ 0 & 1 \end{bmatrix} s_f = \begin{bmatrix} \tilde{R}_f & \tilde{t_f} \end{bmatrix} s_f. \tag{15}$$

Consider a video with a camera in fixed position, consisting of F frames, where in each frame we collect the coordinates of N points $\{p_{f,i}\}_{i,f=1}^{N,F}$ in the $x^C y^C$–plane. This creates a data matrix $X \in \mathbb{R}^{F\times N}$ such that

$$X = \begin{bmatrix} x_{1,1}^C & \cdots & x_{1,N}^C \\ y_{1,1}^C & \cdots & y_{1,N}^C \\ x_{2,1}^C & \cdots & x_{2,N}^C \\ y_{2,1}^C & \cdots & y_{2,N}^C \\ \vdots & \vdots & \vdots \\ x_{F,1}^C & \cdots & x_{F,N}^C \\ y_{F,1}^C & \cdots & y_{F,N}^C \end{bmatrix} = \begin{bmatrix} \tilde{R}_1 & \tilde{t}_1 \\ \tilde{R}_2 & \tilde{t}_2 \\ \vdots & \vdots \\ \tilde{R}_F & \tilde{t}_F \end{bmatrix} \begin{bmatrix} s_1 & \dots & s_n \end{bmatrix} =: MS, \tag{16}$$

Where M describes the motion of the system and S describes its shape.

Note that $M \in \mathbb{R}^{2F\times 4}$ and $S \in \mathbb{R}^{4\times N}$, and $X \in \mathbb{R}^{2F\times N}$. This implies that $\operatorname{rank}(X) \le 4$ since $\operatorname{rank}(X) \le \min\{\operatorname{rank}(M), \operatorname{rank}(S)\}$. That is, the trajectory vectors in X lie on a 4–dimensional subspace of the ambient space $\mathbb{R}^{2F}$.

Now, if X consists of trajectories of feature points on L objects, then $\operatorname{rank}(X) \le 4L$, and trajectory vectors from each object lie on a subspace of dimension at most 4. Thus, we can treat the data as a Subspace Clustering Problem, in which we separate the 4–dimensional subspaces and thus identify which trajectories come from distinct objects.

11.1.2. *Facial Recognition.* Another application which reduces down to a Subspace Clustering Problem comes from Basri and Jacobs [5]. They model what happens when images are taken of faces (or more generally smooth objects) under different lighting conditions, and show using a spherical harmonic expansion, that the images of a given face correspond to an approximately 9-dimensional subspace. Thus if we have many images of different faces under varying illumination, treating them as a Subspace Clustering Problem implies we intend to separate which images are of the same face and which are not.

11.1.3. *Cryo-Electron Microscopy.* Cryo-Electron Microscopy (Cryo-EM) is a technique which won the Nobel Prize in Chemistry in 2017, and is used to determine the 3-D structure of protein molecules. Cryo-EM is a method in which researchers flash-freeze many of the same protein molecule in a substrate, and image them simultaneously via an electron microscope. The fundamental challenges are that the signal obtained is very noisy, and the molecules are all in a random orientation that is unknown to the researchers (though this allows for incredible advances in speed and cost of imaging). Embedded in the reconstruction task is a Subspace Clustering problem (also naturally viewed as orbit reconstruction [4]).

11.2. **Clustering Matrix Methods.** We will focus our attention on one particular method for solving the Subspace Clustering Problem. Our goal is to find a matrix S_X such that

$$\begin{cases} S_X(i,j) \neq 0, & x_i \text{ and } x_j \text{ are in the same subspace} \\ S_X(i,j) = 0, & \text{otherwise.} \end{cases} \tag{17}$$

A matrix which satisfies (17) will be called a **clustering matrix for** X. One of the first approaches to find such a matrix was the Shape Interaction Matrix (SIM) due to Costeira and Kanade [8].

Example 11.2 (Shape Interaction Matrix (SIM))**.** If $X = U_r \Sigma_r V_r$, define

$$\mathrm{SIM}(X) = |V_r V_r^T|^{d_{\max}},$$

where $d_{\max} = \max \dim(S_i)$.

There are some notable issues with this construction. First, one needs an estimate of the number of clusters, k, before forming the shape interaction matrix; this could be done by the elbow method applied to the k–means objective function for varying values of r to form SIM. Second, the SIM is not always a clustering matrix for subspace data [1], but this is easily remedied. Lastly, the SIM is not very robust to noise. To illustrate this fact, consider the illustration in Figure 17.

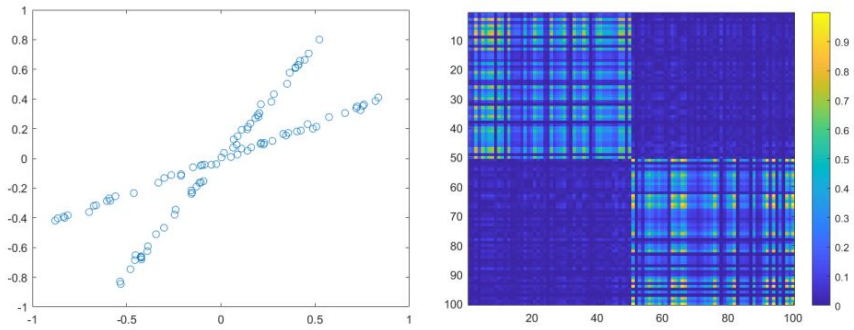

FIGURE 17. (Left) very slightly noisy data from two lines in $\mathbb{R}^2$. (Right) the corresponding SIM.

At first glance, the SIM in Figure 17 seems to do a fairly good job of separating the data into two blocks on the diagonal (which makes sense since there are two subspaces), but the blocks have multiple lines of deep blue cutting through them (which are zero or almost zero entries), making it harder to distinguish them within the matrix. The data was constructed in order from left to right tracing across one line and then the other, so that points near the origin correspond to the center of the two blocks in the right of Figure 17. There may naturally be ambiguity near the origin in subspace clustering, but what is concerning are the deep blue lines at the edges of the block; the presence of these means that the SIM will classify points on opposite ends of the same line as different, which is highly undesirable.

It should be noted that $\mathrm{SIM}(X)$ might only be a clustering matrix for sufficiently "nice" subspace configurations. One of the standard assumptions in the literature is that the intersection of the subspaces is $\{0\}$ and $\dim(S_1+\cdots+S_k) = \dim(S_1)+\cdots+\dim(S_k) \leq \min\{n, m\}$ (such subspaces are called *independent*). Another common assumption is that data within each subspace is *generic*: a collection of points $Y = \{y_1, \ldots, y_n\}$ in a subspace S of dimension d are generic provided any d points in Y are linearly independent. If Y is drawn uniformly at random from S, then the points are essentially guaranteed to

be generic (with probability 1 for those of you who go on to take an advanced probability course).

Let us now present the main theorem concerning clustering matrix solutions to Subspace Clustering.

Theorem 11.3. *Suppose X is drawn from a union of subspaces $\mathscr{U}$ which are independent, and the data within each subspace is generic. Suppose that $X = BY$ where the columns of B form a basis for the column space of X, and moreover the columns of B lie in $\mathscr{U}$. Let $d_{\max} = \max\{\dim(S_i)\}$ and $Q = |Y^T Y|$. Then $Q^{d_{\max}}$ is a clustering matrix for A.*

The proof of this theorem is not given here, but we will briefly indicate the inner workings of it as it connects us back with the graph theory we touched upon earlier. First, we need the definition of the diameter of a graph.

Definition 11.4. *The diameter of a graph G is the maximal distance between any two vertices in G. That is,*

$$\operatorname{diam}(G) := \max\{d_G(u,v) : u, v \in V\},$$

where

$$d_G(u,v) := \min\left\{ \sum_{k=1}^{n} w_{i_k,i_{k+1}} : u = i_1, v = i_n, (i_k, i_{k+1}) \in E \right\}$$

Theorem 11.5. *With the notation and assumptions of Theorem 11.3, suppose that $X = BY$ and $Q = |Y^T Y|$. Up to permutation of the columns of Q,*

$$Q = \begin{bmatrix} A_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & A_m \end{bmatrix}$$

where each A_i is the adjacency matrix of a connected graph of diameter at most $\dim(S_i)$.

Lemma 11.6. *If G has diameter d and A is its adjacency matrix, then A^d is the adjacency matrix of a fully connected graph.*

Combining Theorem 11.5 and Lemma 11.6 yields the conclusion of Theorem 11.3

This concludes our brief discussion of Subspace Clustering, but the interested reader is invited to consult [16] for a broader survey of the problem and its solutions.

12. Numerical Linear Algebra

The goal of this lecture is to describe a few different ways that QR decompositions may be computed numerically, with the ultimate aim of later sketching a numerical approximation to the SVD. Recall that some of the problems we want to solve numerically are,

$$Ax = b,\ Ax = \lambda x, \text{ and } Av = \sigma u.$$

Before we begin, here are several principles to keep in mind when dealing with numerical algorithms:

(1) Matrix-vector multiplication is fast (especially if the matrix is sparse)
(2) Matrix-matrix multiplication is costly (we never want to actually evaluate A^TA or A^n)
(3) Computing determinants is costly.

Suppose that $A \in \mathbb{R}^{n\times n}$, and $x \in \mathbb{R}^n$. Then here are the complexities (number of multiplications) required to evaluate the following quantities. Note that these are complexities of the naïve methods only, and in many cases can be substantially improved. We leave these considerations to a Numerical Analysis course.

Computed Quantity	Cost (number of multiplications in naïve method)
Ax	n^2
AA^T, A^TA, A^2	n^3
$det(A)$	$O(n!)$

12.1. **QR Decomposition: Method 1 (Gram–Schmidt).** The first way we approach QR decomposition is using the Gram-Schmidt process to orthogonalize the columns of our matrix A. The orthonormal columns will become the columns of Q, and as we go through the Gram–Schmidt procedure we will simultaneously produce R. Let $A \in \mathbb{R}^{m\times n}$, and without loss of generality assume $m \geq n$. Write A as $A = \begin{bmatrix} a_1 & \dots & a_n \end{bmatrix}$ where a_i is the i^{th} column of A. Note that $a_1,\dots,a_n$ span Col(A) by definition, and the Gram–Schmidt procedure on these vectors will produce a an orthonormal basis for Col(A). For the first step we have $q_1 = \frac{a_1}{|a_1|}$, then for $i > 1$, we have

$$x_i = a_i - \sum_{j=1}^{i-1} \frac{\langle x_i, q_j\rangle}{|q_j|} q_j,$$
$$q_i = \frac{x_i}{|x_i|}.$$

At the end of the procedure, we define $Q := \begin{bmatrix} q_1 & \cdots & q_n \end{bmatrix}$, and

$$R := \begin{bmatrix} \langle a_1, q_1\rangle & \langle a_2, q_1\rangle & \cdots & \langle a_n, q_1\rangle \\ 0 & \langle a_2, q_2\rangle & \cdots & \langle a_n, q_2\rangle \\ \vdots & \ddots & \ddots & \vdots \\ 0 & \cdots & 0 & \langle a_n, q_n\rangle \end{bmatrix}.$$

Using the properties of matrix multiplication, inner products, and orthogonality gives us $A = QR$, where indeed Q is orthogonal and R is upper triangular. This method is straightforward to implement, but there are some drawbacks to it. In particular, it is numerically unstable and liable to produce a q_i that is nowhere near being orthogonal to some q_j. This gives motivation for finding other ways of computing the QR decomposition, which we will analyze subsequently.

12.2. **QR Decomposition: Method 2 (Column-Pivoted QR).** There is a simple modification to the Gram–Schmidt procedure for producing QR decompositions, which is to add a column-pivoting step. This is common in many algorithms, including those for computing LU decompositions discussed briefly at the beginning of the course.

Algorithm 4: Column-Pivoted (Modified) QR

Input : $A \in \mathbb{R}^{m\times n}$

Output: Orthogonal matrix $Q \in \mathbb{R}^{m\times m}$ and upper triangular matrix $R \in \mathbb{R}^{m\times n}$ such that $A = QR$

$A^{(0)} = A$, $Q^{(0)} = R^{(0)} = []$

for $i = 1 : n$ **do**

$\text{index} = \text{argmax}_\ell \left| A^{(i-1)}_{:\ell} \right|$

$q_i = \frac{A^{(i-1)}_{:\text{index}}}{\left| A^{(i-1)}_{:\text{index}} \right|}$

$Q^{(i)} = \begin{bmatrix} Q^{(i-1)} & q_i \end{bmatrix}$

$r_i = q_i^T A^{(i-1)}$

$R^{(i)} = \begin{bmatrix} R^{(i-1)} \\ r_i \end{bmatrix}$

$A^{(i)} = A^{(i-1)} - q_i r_i$

end

return $Q^{(n)}$, $R^{(n)}$

The Modified QR Algorithm above is typically somewhat more stable than the standard Gram–Schmidt implementation, but still can suffer from errors.

12.3. **QR Decomposition: Method 3 (Householder Reflections).** One of the more common actual implementations of QR in practice is the algorithm we describe here in terms of so-called Householder Reflections.

Definition 12.1. *Let $v \in \mathbb{R}^n$ be a unit vector. Then $H := I - 2vv^T$ is called a **Householder reflection with respect to** v. (Some references call H a Householder reflector.)*

Let us first note that if v and H are as in Definition 12.1, and if $\langle w, v\rangle = 0$, then $Hw = w$. Indeed,

$$Hw = w - 2vv^T w = w - 2v\langle w, v\rangle = w - 0.$$

Now any $x \in \mathbb{R}^n$ can be written as $x = \langle x, v\rangle v + w$ for some w which is orthogonal to v. Hence, for any x, we have

$$\begin{aligned} Hx &= (I - 2vv^T)x \\ &= (I - 2vv^T)(\langle x, v\rangle v + w) \\ &= \langle x, v\rangle v + w - 2vv^T w - 2\langle x, v\rangle vv^T v \\ &= -\langle x, v\rangle v + w. \end{aligned}$$

This gives us an indication of the geometric meaning of Hx given x. Indeed, Hx is the reflection of x across the line orthogonal to v.

To find a QR decomposition via Householder reflections, first choose H_1 such that

$$H_1 a_1 = \begin{bmatrix} |a_1| \\ 0 \\ \vdots \\ 0 \end{bmatrix} = |a_1| e_1,$$

where e_1 is the unit vector $\begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$. Note that we can do this by setting

$$v_1 = a_1 + \operatorname{sgn}(a_{11}) |a_1| e_1, \qquad H_1 = I - 2\frac{v_1 v_1^T}{|v_1|^2},$$

where sgn is the signum function defined via

$$\operatorname{sgn}(x) := \begin{cases} 1, & x \geq 0 \\ 0, & x < 0. \end{cases}$$

We then set $Q_1 = H_1$, and notice that

$$Q_1 A = \begin{bmatrix} a_{11} & * & \cdots & * \\ 0 & & & \\ \vdots & & A_1 & \\ 0 & & & \end{bmatrix}.$$

Now we choose H_2 such that

$$H_2 (A_1)_{:1} = |(A_1)_{:1}| \, e_1,$$

and set

$$Q_2 = \begin{bmatrix} 1 & 0 \\ 0 & H_2 \end{bmatrix},$$

which yields

$$Q_2 Q_1 A = \begin{bmatrix} a_{11} & * & * & \cdots & * \\ 0 & a_{22} & * & \cdots & * \\ 0 & 0 & & & \\ \vdots & \vdots & & A_2 & \\ 0 & 0 & & & \end{bmatrix}.$$

We proceed until we find $Q_1, \ldots, Q_n$ such that $Q_n \ldots Q_1 A = R$, and note that $Q := (Q_n \ldots Q_1)^T$ is orthogonal by construction.

This construction of QR decompositions via Householder reflections is typically more numerically stable than either of the Gram–Schmidt variants discussed before. There is another oft-utilized algorithm using *Givens Rotations*, but we will not discuss this variant here.

13. The Geršgorin Circle Theorem and The Power Method

The goal for this lecture is to learn how to find leading eigenvalues and eigenvectors using the Geršgorin Circle Theorem and the Power Method.

13.1. **Geršgorin Circle Theorem.** Our main theorem of this section is one that tells us where in the complex plane the eigenvalues of a matrix may lie.

Theorem 13.1. *(Geršgorin Circle Theorem) Let $A \in \mathbb{R}^{n\times n}$ and $R_i := \{z \in \mathbb{C} : |z - a_{ii}| \leq \sum_{j=1, j\neq i}^{n} |a_{ij}|\}$, $i = 1, \ldots, n$. Then the eigenvalues of A lie in $\bigcup_{i=1}^{n} R_i$.*

Let's begin with an example of The Geršgorin Circle Theorem.

Example 13.2. Let $A = \begin{bmatrix} 4 & 1 & 1 \\ 0 & 2 & 1 \\ -2 & 0 & 9 \end{bmatrix}$.

Then the following sets describe discs where the eigenvalues of the matrix A lie:

$$R_1 = \{z \in \mathbb{C} : |z - 4| \leq 2\}$$
$$R_2 = \{z \in \mathbb{C} : |z - 2| \leq 1\}$$
$$R_3 = \{z \in \mathbb{C} : |z - 9| \leq 2\}.$$

Note that if the discs are disjoint, then there is one eigenvalue in each disc.

13.2. **Power Method.** Suppose that $A \in \mathbb{R}^{n\times n}$ has eigenvalues $|\lambda_1| > |\lambda_2| \geq |\lambda_3| \geq \cdots \geq |\lambda_n|$ with linearly independent eigenvectors $v_1, \ldots, v_n$. Note that the quantity $|\lambda_1| - |\lambda_2|$ is called the spectral gap. This is an extra assumption, as not all matrices have this.

The goal of the power method is to estimate the leading, largest eigenvalue λ_1 and its associated eigenvector v_1. Before writing the formal algorithm, let us make some notes. To begin, take an initial guess for the eigenvector of λ_1 to be $x^{(0)} = \alpha_1 v_1 + \cdots + \alpha_n v_n$ for some $\alpha_i \neq 0$. In particular, for simplicity we can let $\alpha_i = 1$ for all i. Then $x^{(0)} = v_1 + \cdots + v_n$.

We will then compute

$$x^{(1)} = Ax^{(0)}, \quad x^{(2)} = Ax^{(1)} = A^2x^{(0)}, \quad \ldots \quad x^{(k)} = Ax^{(k-1)} = A^kx^{(0)}.$$

Since $Av_i = \lambda_i v_i$ for all i, we have $A^2v_i = A(Av_i) = A(\lambda_i v_i) = \lambda_i Av_i = \lambda_i^2 v_i$. Similarly, $A^k v_i = \lambda_i^k v_i$.

Now, for $x^{(0)} = v_1 + \cdots + v_n$, we have

$$\begin{aligned} x^{(k)} &= A^k x^{(0)} \\ &= A^k(v_1 + \cdots + v_n) \\ &= \lambda_1^k v_1 + \cdots + \lambda_n^k v_n \\ &= \lambda_1^k \left(v_1 + \left(\frac{\lambda_2}{\lambda_1}\right)^k v_2 + \cdots + \left(\frac{\lambda_n}{\lambda_1}\right)^k v_n \right). \end{aligned}$$

Note that, by assumption, $\frac{\lambda_i}{\lambda_1} < 1$ for $i = 2, \ldots, n$. Thus $(|\frac{\lambda_i}{\lambda_1}|)^k \to 0$ as $k \to \infty$. For simplicity in what follows, we write $x^{(k)} = \lambda_1^k(v_1 + \varepsilon^{(k)})$.

13.2.1. *Aside: Linear Functionals.* A linear functional on $\mathbb{R}^n$ is a map $\phi : \mathbb{R}^n \to \mathbb{R}$ such that for all $\alpha, \beta \in \mathbb{R}$ and for all $x, y \in \mathbb{R}^n$, $\phi(\alpha x + \beta y) = \alpha\phi(x) + \beta\phi(y)$. The following is a simple example of a linear functional. Let $\phi(x) = x_1$. Then

$$\phi\left(\alpha \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} + \beta \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}\right) = \phi\left(\begin{bmatrix} \alpha x_1 + \beta y_1 \\ \vdots \\ \alpha x_n + \beta y_1 \end{bmatrix}\right) = \alpha x_1 + \beta y_1 = \alpha\phi(x_1) + \beta\phi(y_1).$$

A basic, but important fact about linear functionals for our purposes is that if $\epsilon^{(k)} \to 0$ as $k \to \infty$, then $\phi(\epsilon^{(k)}) \to 0$ as $k \to \infty$ as well (i.e. linear functionals preserve convergence).

Going back to the power method, let's take another look at $x^{(k)}$. If ϕ is any linear functional, then

$$\phi(x^{(k)}) = \phi\left(\lambda_1^k v_1 + \lambda_1^k \varepsilon^{(k)}\right) = \lambda_1^k \left(\phi(v_1) + \phi(\varepsilon^{(k)})\right).$$

We cannot directly take the limit as $k \to \infty$ in the above expression because it could be that $\lambda_1^k \to \infty$ (if $|\lambda_1| > 1$) while $\phi(\varepsilon^{(k)}) \to 0$, and we would need to know something about the rates of convergence to say anything about the limit of their product. To get around this, we consider the quotient

$$r_k = \frac{\phi(x^{(k+1)})}{\phi(x^{(k)})} = \frac{\lambda_1^{k+1}\left(\phi(v_1) + \phi(\varepsilon^{(k+1)})\right)}{\lambda_1^k\left(\phi(v_1) + \phi(\varepsilon^{(k)})\right)} = \lambda_1 \frac{(\phi(v_1) + \phi(\varepsilon^{(k+1)})}{(\phi(v_1) + \phi(\varepsilon^{(k)})}.$$

There is no problem taking a limit now, and we can see that

$$\lim_{k\to\infty} r_k = \lambda_1.$$

Our goal was to find λ_1, and the above analysis gives us an outline for how to do it.

Finally, it should be noted that the convergence rate of r_k is $|\frac{\lambda_2}{\lambda_1}|^k$, meaning that $r_k \leq \lambda_1 + C|\frac{\lambda_2}{\lambda_1}|^k$ for sufficiently large k for some absolute constant C (this statement is often written $r_k = \lambda_1 + O(|\frac{\lambda_1}{\lambda_2}|^k)$).

13.2.2. *Algorithm.* To better understand the power method, we consider the algorithm in some detail. In the implementation, a linear functional ϕ is chosen for the algorithm and used for all inputs.

The algorithm takes as inputs a matrix $A \in \mathbb{R}^{n\times m}$, an initial guess $x^{(0)} \in \mathbb{R}^n$, a maximum number of allowed iterations, $N_{\max}$, and an error tolerance, TOL. As before, a simple initialization is to choose $x^{(0)}$ to be the all-ones vector, $\mathbb{1}$. With this set of inputs, the algorithm will continue executing until the change enacted in one iteration (measured by the ratio of the linear functional applied to the old and new guess for x) is smaller than the tolerance, or the maximum number of iterations is reached – whichever occurs first.

The Power Method algorithm is described in pseudocode in Algorithm 5.

In the code above, for an iteration k, $x = x^{(k)}$ and $y = x^{(k+1)}$. Likewise, $r_{\text{old}} = r_k$ and $r_{\text{new}} = r_{k+1}$. To make the algorithm a bit clearer, let us go through a simple example.

Algorithm 5: The Power Method

Input : Matrix $A \in \mathbb{R}^{n\times m}$ with $|\lambda_1| > |\lambda_2|$, initial guess $x^{(0)} \in \mathbb{R}^n$, max number of iterations $N_{\max}$, error tolerance TOL

Output: Estimates of the first eigenvalue/eigenvector pair (λ_1, v_1)

$k = 1$

$x = \dfrac{x^{(0)}}{\|x^{(0)}\|_\infty}$

$r_{\text{old}} = 0$

while $k \le N_{\max}$ **do**

  $y = Ax$

  $r_{\text{new}} = \dfrac{\phi(y)}{\phi(x)}$

  $x = \dfrac{y}{\|y\|_\infty}$

  if $|r_{\text{new}} - r_{\text{old}}| < \text{TOL}$ **then**

    return (r_{new}, x)

  else

    $k = k + 1$

    $r_{\text{old}} = r_{\text{new}}$

  end

end

return r_{new}, x

13.2.3. *Power Method Example.* Let $A = \begin{bmatrix} -4 & 14 & 0 \\ -5 & 13 & 0 \\ -1 & 0 & 2 \end{bmatrix}$, and choose $\phi(x) = x_1$ and $x^{(0)} = \mathbb{1}$. In this case, the actual eigenvalues of A are $\lambda_1 = 6, \lambda_2 = 3$, and $\lambda_3 = 1$, with the leading eigenvector $v_1 = \begin{bmatrix} 1 \\ \frac{5}{7} \\ \frac{-1}{4} \end{bmatrix} \approx \begin{bmatrix} 1 \\ 0.714 \\ -0.25 \end{bmatrix}$.

We'll walk through the first 3 iterations, using numbered subscripts/superscripts to represent the current iteration. Decimal values are used for simpler visual comparison to the approximate decimal values in v_1.

Iteration 1: $y_1 = Ax^{(0)} = \begin{bmatrix} 10 \\ 8 \\ 1 \end{bmatrix}$, $r_1 = \frac{10}{1} = 10$, $x^{(1)} = \begin{bmatrix} 1 \\ 0.8 \\ 0.1 \end{bmatrix}$

Iteration 2: $y_2 = Ax^{(1)} = \begin{bmatrix} 7.2 \\ 5.4 \\ -0.8 \end{bmatrix}$ $r_2 = \frac{7.2}{1} = 7.2$ $x^{(2)} = \begin{bmatrix} 1 \\ 0.75 \\ -0.111 \end{bmatrix}$

Iteration 3: $y_3 = Ax^{(2)} = \begin{bmatrix} 6.5 \\ 4.75 \\ -0.25 \end{bmatrix}$ $r_3 = \frac{6.5}{1} = 6.5$ $x^{(3)} = \begin{bmatrix} 1 \\ 0.77 \\ -0.188 \end{bmatrix}$.

Even these three iterations are enough to see that the value of r is approaching λ_1 and the vector x is approaching v_1.

The power method is widely used for several reasons: it is simple to implement; there is no matrix-matrix multiplication (only one matrix-vector multiplication, a vector norm, and one vector-scalar multiplication per iteration); it is computationally non-intensive. It is also spatially non-intensive, requiring only the storage of A, vectors x and y, and the ratio r.

13.2.4. *Other Eigenvalues.* As written, the power method is a useful and computationally efficient way to compute the first eigenvalue of A. However, what if we want to compute other eigenvalues? To address this problem, we will first make some preliminary observations:

(1) If A is invertible and has eigenvalues $\lambda_1 \dots \lambda_n$, then A^{-1} has eigenvalues $\frac{1}{\lambda_1} \dots \frac{1}{\lambda_n}$ with the same eigenvectors.
(2) Suppose $q \in \mathbb{C}$ is not an eigenvalue of A. Then $\det(A - qI) \neq 0$, and $A - qI$ has eigenvalues $\lambda_i - q$ for $i = 1, \dots, n$.
(3) The eigenvalues of $B = (A - qI)^{-1}$ are $\frac{1}{\lambda_1 - q}, \dots, \frac{1}{\lambda_n - q}$. Note that the largest eigenvalue of B is the eigenvalue of A which is **closest** to q.

No we may use these observations to conclude that if we can judiciously choose values of q, we may estimate other eigenvalues of A by running the power method on $B = (A - qI)^{-1}$, the result of which will be the λ_i closest to q. This observation is the key to the Inverse Power Method.

13.2.5. *Inverse Power Method.* The Inverse Power Method is named because it is the power method applied to the inverse of a matrix related to the initial A. Specifically, the inverse power method runs Algorithm 5 on $(A - qI)^{-1}$ for a given value of q to find $\frac{1}{\lambda_i - q}$, then calculates and returns λ_i

At a first glance, there are two issues with this approach. One issue is that the inverse power method requires that $|\lambda_i| > |\lambda_{i+1}|$. Another pressing issue is that we must be able to estimate q. However, this problem is easily solved. Recall that the Geršgorin Circle Theorem allows a simple method for determining the range of values an eigenvalue can take. We can choose any point in one of the discs (in particular, its center) as q to estimate an eigenvalue which lies in that disc.

Another issue is one of computational cost. To run the power method on $(A - qI)^{-1}$, we must solve $y^{(k)} = (A - qI)^{-1} x^{(k-1)}$ for $y^{(k)}$. This ostensibly requires inverting $A - qI$, which could be quite expensive for a large matrix A. Instead, we can use Gaussian elimination to solve $(A - qI)^{-1} y^{(k)} = x^{(k-1)}$ instead, reducing the computational cost.

With these two issues resolved, the inverse power method becomes a powerful extension of the power method to compute any or all eigenvalues of a large matrix.

14. Numerically Computing the SVD

We end our brief tour of numerical Linear Algebra by sketching one method of approximating the SVD of a matrix. This section will contain a brief sketch of an SVD solver due to Golub and Kahane. Our exposition was heavily influenced by [7].

Step 1: Bidiagonalize $A^T A$ implicitly via Algorithm 6.

Algorithm 6: Bidiagonalization algorithm

Input : Matrix $A \in \mathbb{R}^{m\times n}$

Output: Orthogonal matrices $U \in \mathbb{R}^{m\times m}$, $V \in \mathbb{R}^{n\times n}$, and $B \in \mathbb{R}^{m\times n}$ which is upper bidiagonal (i.e., B only has nonzero entries on its diagonal and the diagonal starting with b_{12}) such that $A = UBV^T$.

Initialize $B = A$, $U = I_{m\times m}$, $V = I_{n\times n}$

for *i=1* **to** n **do**

Find a Householder reflector H_k such that $H_k \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b_{k-1,k} \\ \vdots \\ b_{mk} \end{bmatrix} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ b_{k-1,k} \\ \pm\sqrt{\sum_{i=k}^{m} b_{ik}^2} \\ \vdots \\ 0 \end{bmatrix}$

$B = H_k B$

$U = U H_k$

if $k \le n-2$ **then**

find householder reflector P_{k+1} such that

$$\begin{bmatrix} 0 & \cdots & 0 & b_{kk} & \cdots & b_{kn} \end{bmatrix} P_{k+1} = \begin{bmatrix} 0 & \cdots & 0 & b_{kk} & \pm\sqrt{\sum_{j=k+1}^{n} b_{kj}^2} & 0 & \cdots & 0 \end{bmatrix}$$

$B = B P_{k+1}$

$V = P_{k+1} V$

end

end

return U, B, V

Step 2: Get rid of small off-diagonal entries of B (i.e. those which are numerically equal to the near by diagonal entry of whose diagonal entry is already 0)

Step 3: Solve several 2×2 SVDs explicitly.

Then repeat steps 2 and 3 until B is diagonal and $UBV^T = A$.

15. Least Squares and Pseudoinverses

The goal of this lecture is to examine the Least Squares problem, and its solution in terms of various factorizations we have discussed, such as the QR decomposition and the SVD. We will derive a closed-form expression for the least squares solution to a general overdetermined system of equations, which will lead us to a formal definition of the Moore–Penrose pseudoinvers of a matrix.

Recall that if $A \in \mathbb{R}^{m\times n}$ with $m \geq n$, then $Ax = b$ either has 0 or 1 solution. Least Squares attempts to answer the question: if there are no solutions, what is a good approximate solution to $Ax = b$?

That is, we want to solve the following optimization problem: given $A \in \mathbb{R}^{m\times n}$ and $b \in \mathbb{R}^m$, find

$$\widehat{x} = \operatorname*{argmin}_{x\in\mathbb{R}^n} |Ax - b|^2 . \tag{18}$$

We can take the argument in (17) as a function $f : \mathbb{R}^n \to \mathbb{R}$, and note that

$$f(x) := |Ax - b|^2 = \sum_{i=1}^{m} \left(\sum_{j=1}^{n} a_{ij}x_j - b_i \right)^2 .$$

From Calculus, we know that a necessary condition for $\widehat{x}$ to be a minimizer of f is that $\frac{\partial f}{\partial x_k} = 0$ for $k = 1, 2, \ldots, n$. This observation leads to the **Normal Equations:**

$$A^T A\widehat{x} = A^T b.$$

Indeed, the normal equations may be derived as follows.

First, for a given k, we compute the partial derivative of f with respect to x_k. We will suppress the beginning and ends of indices in summations for ease of reading, but all sums over i are from 1 to m and all sums over j are from 1 to n. Back to your multivariate calculus days, note that for a given k, we have

$$\begin{aligned}
\frac{\partial}{\partial x_k} \sum_i \left(\sum_j a_{ij}x_j - b_i \right)^2 &= \sum_i \frac{\partial}{\partial x_k} \left(\sum_j a_{ij}x_j - b_i \right)^2 \\
&= 2\sum_i \left(\sum_j a_{ij}x_j - b_i \right) a_{ik} \\
&= 2\sum_i \sum_j a_{ij}a_{ik}x_j - 2\sum_i a_{ik}b_i .
\end{aligned}$$

Setting this equal to 0, we may ignore the factor of 2.

Now, since we must have $\frac{\partial}{\partial x_k}|Ax - b|^2 = 0$ for all k, we find

$$\sum_i a_{ik} \left(\sum_j a_{ij}x_j - b_i \right) = \sum_i a_{ik} \left((Ax)_i - b_i \right) = (A^T Ax)_k - (A^T b)_k = 0.$$

Hence

$$A^T Ax = A^T b$$

as required.

We will now proceed to make a series of observations about the solutions to the Least Squares problem. Our goal here is to understand the normal equations in terms of the various matrix factorizations we have discussed thus far.

15.1. **Observation 1:** If A has linearly independent columns, then A^TA has full rank, and thus is invertible. Consequently, $A^TA\widehat{x} = A^Tb$ implies that

$$\widehat{x} = (A^TA)^{-1}A^Tb. \tag{19}$$

15.2. **Observation 2:** If A has linearly independent columns, and $A = QR$, then Q has orthonormal columns by construction, and R is invertible. We did not discuss this feature of R previously, but the proof is that $Q^TA = Q^TQR = R$, so $\text{rank}(R) \leq \text{rank}(A) = n$, whereas on the other hand, $n = \text{rank}(A) = \text{rank}(QR) \leq \text{rank}(R)$, and thus $\text{rank}(R) = n$.

Let's have another look at the normal equations in this case. Now the normal equations are

$$R^TQ^TQR\widehat{x} = R^TQ^Tb.$$

This yields $R^TR\widehat{x} = R^TQ^Tb$ by the fact that Q has orthonormal columns. Using the observation that R is invertible, we have $\widehat{x} = R^{-1}(R^T)^{-1}R^TQ^Tb$, whence

$$\widehat{x} = R^{-1}Q^Tb. \tag{20}$$

15.3. **Observation 3:** If A has linearly independent columns and $A = U\Sigma V^T$, then $\Sigma = \begin{bmatrix}\Sigma_n \\ 0\end{bmatrix}$, and the normal equations become

$$V\Sigma^TU^TU\Sigma V^T\widehat{x} = V\Sigma^TU^Tb.$$

Now we use orthogonality of U and the fact that $\Sigma^T\Sigma = \Sigma_n^2$ to see that this becomes $V\Sigma_n^2V^T\widehat{x} = V\Sigma^TU^Tb$, which becomes $\Sigma_n^2V^T\widehat{x} = \Sigma^TU^Tb$ by multiplying on the left by V^T.

Next, since $\Sigma_n = \text{diag}(\sigma_1, \ldots, \sigma_n)$, none of whose diagonal entries are 0, we have that Σ_n is invertible. Multiplying both sides by $\Sigma_n^{-2} := (\Sigma_n^{-1})^2$ yields

$$V^T\widehat{x} = \Sigma_n^{-2}\Sigma^TU^Tb = \Sigma_n^{-2}\begin{bmatrix}\Sigma_n & 0\end{bmatrix}U^Tb = \begin{bmatrix}\Sigma_n^{-1} & 0\end{bmatrix}U^Tb.$$

Finally, multiplying both sides by V gives us

$$\widehat{x} = V\begin{bmatrix}\Sigma_n^{-1} & 0\end{bmatrix}U^Tb. \tag{21}$$

15.4. **The General Case.** Finally, since the previous observations made critical use of the assumption that A had full column rank, we now turn to the general case in which we make no assumption on $\text{rank}(A)$. Still, suppose that $A = U\Sigma V^T$, $\text{rank}(A) = k$, and let us consider the form of the minimization problem (18) directly. The objective function becomes

$$\begin{aligned}\min_x |Ax - b|^2 &= \min_x |U\Sigma V^Tx - b|^2 \\ &= \min_x |\Sigma V^Tx - U^Tb|^2,\end{aligned}$$

where the equality is due to the fact that $|U^Ty| = |y|$ for any y on account of orthogonality of U.

Now since V^T is a bijection from $\mathbb{R}^n \to \mathbb{R}^n$, any $w \in \mathbb{R}^n$ can be realized as V^Tx for some x. So we recast this problem as

$$\begin{aligned}\min_x |\Sigma V^Tx - U^Tb|^2 &= \min_w |\Sigma w - U^Tb|^2 \\ &= \min_w (\Sigma w - U^Tb)^T(\Sigma w - U^Tb) \\ &= \min_w (w^T\Sigma^T - b^TU)(\Sigma w - U^Tb)\end{aligned}$$

$$
\begin{aligned}
&= \min_w w^T \Sigma^T \Sigma w - b^T U \Sigma w - w^T \Sigma^T U^T b + b^T U U^T b \\
&= \min_w |\Sigma w|^2 - 2\left\langle \Sigma w, U^T b \right\rangle + |U^T b|^2 \\
&= \min_w \sum_{i=1}^{k} (\sigma_i w_i)^2 - 2\sum_{i=1}^{k} \sigma_i w_i (U^T b)_i + \sum_{i=1}^{n} (U^T b)_i^2 \\
&= \min_w \sum_{i=1}^{n} \left(\sigma_i w_i - (U^T b)_i\right)^2.
\end{aligned}
$$

Since each term in the summation is nonnegative, the minimum value is easily seen to be achieved by setting

$$w_i = \frac{(U^T b)_i}{\sigma_i}, \quad i = 1, \ldots, k.$$

Now we are free to set $w_{k+1}, \ldots, w_n$ to whatever we like. To obtain the smallest norm w which is a minimizer of the objective function above, we can simply take $w_{k+1} = \cdots = w_n = 0$. Now we have

$$w = \begin{bmatrix} \Sigma_k^{-1} & 0 \\ 0 & 0 \end{bmatrix} U^T b = V^T \widehat{x},$$

whereby we simply define $\Sigma^\dagger$ to be the $n \times m$ matrix with entries $\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_k}$ along its main diagonal, and we have

$$\widehat{x} = V \Sigma^\dagger U^T b. \tag{22}$$

Now we may formally define the solution to this, which carries the name Moore–Penrose pseudoinverse to honor the two authors who first discovered it.

Definition 15.1. *If $A = U\Sigma V^T$, then its Moore–Penrose Pseudoinverse is $A^\dagger := V\Sigma^\dagger U^T$, where $\Sigma^\dagger \in \mathbb{R}^{n \times m}$ has diagonal entries $\frac{1}{\sigma_1}, \ldots, \frac{1}{\sigma_{rank(A)}}$.*

With this definition, the Least Squares solution to $A^T A\widehat{x} = A^T b$ is $\widehat{x} = A^\dagger b$.

16. The Moore–Penrose Pseudoinverse

For this lecture, we will further discuss the pseudoinverse we discovered by solving the Least Squares problem last time. In particular, we are most interested in how it acts on the four fundamental subspaces introduced early on in the course, and in proving various properties of the pseudoinverse.

Theorem 16.1. *Let $A \in \mathbb{R}^{m\times n}$. Then its Moore–Penrose pseudoinverse is the unique matrix $A^\dagger \in \mathbb{R}^{n\times m}$ which satisfies*

(1) $AA^\dagger A = A$
(2) $A^\dagger AA^\dagger = A^\dagger$
(3) $(AA^\dagger)^T = AA^\dagger$
(4) $(A^\dagger A)^T = A^\dagger A$.

16.1. **Proof of Theorem 16.1.** Before giving the proof of the theorem, we need to collect some useful observations about some of the matrices involved. This also gives us good practice with computations involving pseudoinverses.

Lemma 16.2. *Let $A \in \mathbb{R}^{m\times n}$, and suppose that* $\operatorname{rank}(A) = k$ *and that $A = U_k\Sigma_k V_k^T$ is its truncated SVD. Then*

(1) $AA^\dagger = U_k U_k^T$
(2) $A^\dagger A = V_k V_k^T$.

Proof. (1): For now, the easiest way to see this is by using block matrix multiplication.

$$\begin{aligned}
AA^\dagger &= U\Sigma V^T V\Sigma^\dagger U^T \\
&= \begin{bmatrix} U_k & U_{m-k}\end{bmatrix}\begin{bmatrix}\Sigma_k & 0\\ 0 & 0\end{bmatrix} V^T V \begin{bmatrix}\Sigma_k^{-1} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix} U_k^T \\ U_{m-k}^T\end{bmatrix} \\
&= \begin{bmatrix} U_k & U_{m-k}\end{bmatrix}\begin{bmatrix}\Sigma_k & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix}\Sigma_k^{-1} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix} U_k^T \\ U_{m-k}^T\end{bmatrix} \\
&= \begin{bmatrix} U_k & U_{m-k}\end{bmatrix}\begin{bmatrix}I_{k\times k} & 0\\ 0 & 0\end{bmatrix}\begin{bmatrix} U_k^T \\ U_{m-k}^T\end{bmatrix} \\
&= \begin{bmatrix} U_k & 0\end{bmatrix}\begin{bmatrix} U_k^T \\ U_{m-k}^T\end{bmatrix} \\
&= U_k U_k^T.
\end{aligned}$$

The third line used the fact that $V^T V = I$, and we note that $\Sigma_k = \operatorname{diag}(\sigma_1,\dots,\sigma_k)$ is invertible since $\operatorname{rank}(A) = k$, so $\Sigma^\dagger$ has the form we have written above. The rest is seen by carrying out the block matrix multiplication.

(2): This follows exactly the same line of reasoning as part (1), just interchange the roles of U and V. □

Now, using Lemma 16.2, we first show that the Moore–Penrose pseudoinverse indeed satisfies the four conditions of Theorem 16.1. To see the first condition, note that

$$AA^\dagger A = U_k U_k^T U_k \Sigma_k V_k^T = U_k\Sigma_k V_k^T = A,$$

where the second equality stems from the fact that U_k has orthonormal columns which implies $U_k^T U_k = I$.

Condition (2) follows similarly:

$$A^\dagger AA^\dagger = V_k V_k^T V_k \Sigma_k^\dagger U_k^T = V_k \Sigma_k^\dagger U_k^T = A^\dagger,$$

where we used the fact that V_k has orthonormal columns this time.

Conditions (3) and (4) follow directly from Lemma 16.2 because evidently $U_k U_k^T$ and $V_k V_k^T$ are symmetric.

The only remaining issue is the proof of uniqueness. To wit, suppose that $A_1^\dagger$ and $A_2^\dagger$ both satisfy conditions (1)–(4) for some fixed, but arbitrary, A. First, note that

$$AA_1^\dagger = (AA_2^\dagger A)A_1^\dagger = (AA_2^\dagger)^T (AA_1^\dagger)^T = (A_2^\dagger)^T A^T (AA_1^\dagger)^T = (A_2^\dagger)^T (AA_1^\dagger A)^T = (A_2^\dagger)^T A^T = AA_2^\dagger.$$

The first equality used condition (1), the second used (3), and the last equality again used (3).

By a similar argument, we obtain $A_1^\dagger A = A_2^\dagger A$. We put these together to obtain

$$A_1^\dagger = A_1^\dagger AA_1^\dagger = A_1^\dagger AA_2^\dagger = A_2^\dagger AA_2^\dagger = A_2^\dagger.$$

We have concluded that $A_1^\dagger = A_2^\dagger$, and hence is the unique matrix satisfying the required conditions, and the proof is complete. □

16.2. **Properties of the Moore–Penrose Pseudoinverse.**

Proposition 16.3. *If $A \in \mathbb{R}^{n\times n}$ is invertible, then $A^\dagger = A^{-1}$.*

Proof. Note that A^{-1} satisfies all of the conditions of Theorem 16.1, hence by uniqueness, $A^\dagger = A^{-1}$. □

Proposition 16.4. *For any $A \in \mathbb{R}^{m\times n}$, $(A^T)^\dagger = (A^\dagger)^T$.*

Proof. This may be proven by direct verification of the four necessary properties of the Moore–Penrose pseudoinverse. □

Recall that for invertible matrices, we have $(AB)^{-1} = B^{-1}A^{-1}$, which one can verify by direct computation easily. However, in general, $(AB)^\dagger \neq B^\dagger A^\dagger$. Equality here occurs only in special scenarios. To show this, we will appeal to a simple example.

Example 16.5. There exist $A, B \in \mathbb{R}^{n\times n}$ such that $(AB)^\dagger \neq B^\dagger A^\dagger$. Indeed, let

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 2 \\ 3 & 2 \end{bmatrix}.$$

Then

$$A^\dagger = \begin{bmatrix} \frac{1}{2} & 0 \\ \frac{1}{2} & 0 \end{bmatrix}, \quad B^\dagger = \begin{bmatrix} -\frac{1}{2} & \frac{1}{2} \\ \frac{3}{4} & -\frac{1}{4} \end{bmatrix},$$

and

$$B^\dagger A^\dagger = \begin{bmatrix} 0 & 0 \\ \frac{1}{4} & 0 \end{bmatrix},$$

whereas

$$(AB)^\dagger = \begin{bmatrix} 4 & 4 \\ 0 & 0 \end{bmatrix}^\dagger = \begin{bmatrix} \frac{1}{8} & 0 \\ \frac{1}{8} & 0 \end{bmatrix}.$$

We see by inspection that the results are not the same.

We end with a **caution:** pseudoinverses are not stable under small perturbations; i.e., $\|(A+E)^\dagger - A^\dagger\|_2$ can be arbitrarily large even for arbitrarily small E.

Example 16.6. Let

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad E = \begin{bmatrix} 0 & 0 \\ 0 & \varepsilon \end{bmatrix},$$

for some small $\varepsilon > 0$. Recalling how to find pseudoinverses of diagonal matrices, we have

$$A^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}, \qquad (A+E)^\dagger = \begin{bmatrix} 1 & 0 \\ 0 & \frac{1}{\varepsilon} \end{bmatrix}.$$

Thus,

$$\|(A+E)^\dagger - A^\dagger\|_2 = \left\| \begin{bmatrix} 0 & 0 \\ 0 & \frac{1}{\varepsilon} \end{bmatrix} \right\|_2 = \frac{1}{\varepsilon}.$$

Now for very small ε, $\frac{1}{\varepsilon}$ is extremely large, which proves our claim of instability. Note that the same instability occurs for inverses of matrices as well.

17. CUR Decompositions and Column Subset Selection

We mentioned during our prolonged discussion of matrix factorizations that not all bases of the column space of a matrix are created equally, and in some circumstances one factorization is preferred over another. In this section, we highlight factorizations of the form $A = CX$ where C is a column submatrix of A; that is, we represent A in terms of actual columns of A. For a full, concise exposition of this task, I invite the reader to consult [10]; much of the material here is taken from there, but given that I wrote it, I trust the reader will forgive me.

17.1. **Column Subset Selection.**

18. Johnson–Lindenstrauss Embeddings

Now we will turn to a major new topic which will take us through much of (if not all of) the rest of the semester.

19.1. **The Problem Setup.** The input in any supervised learning problem is a set of labeled data. To set some terminology, let Ω be our **data space**, and let Λ be our **label space**.

For example, in simple binary classification tasks, we could have $\Omega = \mathbb{R}^n$ and $\Lambda = \{1,2\}$. More concretely, suppose $\Omega = \mathbb{R}^n$ for large n, and the data we have collected are black and white images of dogs or cats with n pixels; then our label set could be $\Lambda = \{\text{dog, cat}\}$ (which we could of course arbitrarily map to $\{1,2\}$ for convenience of carrying out everything with numbers).

We assume we are given a set $\mathscr{X}_L = \{(x_i^L, y_i)\}_{i=1}^N \subset \Omega \times \Lambda$ of **labelled data**. Additionally, we have another set $\mathscr{X}_U = \{x_i^U\}_{i=1}^m \subset \Omega$ of **unlabelled data**.

Our goal is to find a good labelling for the unlabelled points $\mathscr{X}_U$. The issue is: what do we mean by, and how do we measure how "good" a labelling is? To treat this issue really well would take an entire course, and require a hefty dose of probability, which was not formally a prerequisite for this course. Thus we will gloss over a lot of the details, but will nonetheless give some intuition as we go along. For those of you interested in further reading (though at a much higher level) we suggest the two sources that heavily influenced our exposition here: [11, 14]

Let us call our **labelling function**

$$f : \Omega \to \Lambda.$$

We will discuss how to find such a labelling function later, but at present, suppose we have one in hand. Then we will quantify how good or bad our labelling function via a **loss function**

$$L : \Omega \times \Lambda \times \Lambda \to [0, \infty).$$

19.2. **A detour into Probability.** Okay, we put off our discussion of probability as long as we could, but now we need to go there. For those of you without the background of probability, simply read this section for whatever you can get out of it, and don't worry about the rest. We will not strictly use any of the material here for what comes later, but it nonetheless is the reasoning for our future deliberations and so we include it here.

For simplicity, let us suppose that our labelled data points $\mathscr{X}_L$ are drawn independently from an (unknown) probability distribution $P(x,y)$ on the product space $\Omega \times \Lambda$. In particular, a point (x_i^L, y_i) is generated by first drawing x_i^L from P_Ω (the marginal probability on Ω) and y_i is drawn from the conditional probability distribution on Λ, $P(\cdot|x_i)$.

Given a loss function L, a labelling function f, and *full knowledge* to the probability distribution P, we could assess the **risk** of choosing the labelling function f as

$$\mathscr{R}_{L,P}(f) := \int_{\Omega\times\Lambda} L(x,y,f(x))\,dP(x,y) = \int_\Omega \int_\Lambda L(x,y,f(x))\,dP(y|x)\,dP_\Omega(x).$$

Note that since the loss function is nonnegative, this integral is also nonnegative, and so this risk is estimating how large the loss function is over the whole probability space. Our path is clear at this point: we would like to find the labelling function $f^* : \Omega \to \Lambda$ which is a solution to the minimization problem:

$$\mathscr{R}^*_{L,P} := \inf_{f:\Omega\to\Lambda} \mathscr{R}_{L,P}(f).$$

Of course the problem with this is that we *do not* have full knowledge of P. What we do have are discrete, independent samples of points in $\Omega \times \Lambda$. So from these, we can compute the **empirical risk** of our labelling function via

$$\mathscr{R}_{L,\mathscr{X}_L}(f) := \frac{1}{N}\sum_{i=1}^{N} L(x_i, y_i, f(x_i)).$$

This is nothing but the average loss over the training data for a given labelling function. One option at this point is to try to find

$$\mathscr{R}^*_{L,\mathscr{X}} = \inf_{f:\Omega\to\Lambda} \mathscr{R}_{L,\mathscr{X}_L}(f)$$

and hope that the error

$$\mathscr{R}^*_{L,P} - \mathscr{R}^*_{L,\mathscr{X}_L}$$

is small.

One final bit of complication; the infimum above is taken over all possible labelling functions f, which is an enormous class of functions, even for discrete Ω and Λ. Thus, we often consider labelling functions which are part of some fixed class of functions, $\mathscr{F}$. In this case, we denote the minimizer to the empirical risk over this class of functions via

$$\mathscr{R}^*_{L,\mathscr{X}_L,\mathscr{F}} := \inf_{f\in\mathscr{F}} \mathscr{R}_{L,\mathscr{X}_L}(f).$$

19.3. **Loss Function Examples.** Now let us be concrete and give some examples of loss functions and labelling functions.

19.3.1. ***0–1 Loss***. In a classification task in which $\Lambda = \{-1, 1\}$ – i.e., we wish to distinguish between two different classifications for input vectors, we can define a simple 0–1 loss function via

$$L_{0-1}(x, y, f(x)) := \begin{cases} 0, & f(x) = y \\ 1, & f(x) \neq y. \end{cases}$$

In other words, there is no loss if the label is correct, but a constant loss of 1 if the label is wrong.

The benefit of this loss function is that it is easy to implement and understand, but the downside is that it makes optimization extremely difficult as it is a nonconvex function. We will not delve into this, but the basic idea that you should keep in mind for now is that convex functions can be readily optimized over, whereas nonconvex functions cannot (convex here is synonymous with "concave up" used in many Calculus textbooks).

19.3.2. ***Hinge Loss***. Many times it is useful to take a nonconvex function and attempt to make it convex in some way. This changes the behavior of any optimization problem significantly, but nonetheless is often a good workaround. (Note that we have seen this idea before in our discussion of the fact that Spectral Clustering is a convex relaxation of certain graph cut problems.) With that in mind, one way to convexify the 0–1 loss function is as follows; the *hinge loss* is defined to be

$$L_{\text{hinge}}(x, y, f(x)) := \max\{0, 1 - yf(x)\}.$$

Note that here, if $f(x) = y$, then the loss is 0, but if $(y, f(x)) = (-1, 1)$ or $(1, -1)$, then the hinge loss is 2 in both cases. It should be noted that often, we actually allow $f : \Omega \to \mathbb{R}$ rather than $f : \Omega \to \Lambda$ to allow us to more easily solve the optimization problems necessary to compute (or estimate) the labelling function which minimizes empirical

risk. In this case, we would allow $f(x)$ to take values on $\mathbb{R}$, and the loss function would be between 0 and 2 in some cases (e.g., if $y = 1$ and $f(x) = 0.5$).

19.4. **Labelling Function Examples.** Throughout the rest of our tour through supervised learning, we will focus a lot on how to choose – or in Machine Learning parlance *learn* – the labelling function f from the training data $\mathscr{X}_L$. Eventually we will cover k–nearest neighbor classifiers, Support Vector Machines, and Neural Networks (among possibly others). We will go into each of these in detail, but for now, we just give some simple examples.

19.4.1. ***A useless, but valid labelling function.*** Given our training data $\mathscr{X}_L = \{(x_i, y_i\}_{i=1}^N$, nothing stops us from simply assigning the following labelling function:

$$f(x) = \begin{cases} y_i, & x = x_i \in \mathscr{X}_L \\ 1, & x \notin \mathscr{X}_L. \end{cases}$$

That is, we label all the training data correctly, and simply guess at any new point that is not in the training data. In this case, we will have 100% accuracy on our training set, but our labelling will not generalize at all to new data. We could easily allow f to be random by choosing $f(x)$ from the uniform distribution on Λ for $x \notin \mathscr{X}_L$. This means that for unlabelled points, we simply randomly guess at the label, and is obviously problematic. This labelling function, among other flaws, does not utilize any knowledge we could gain from the training data.

19.4.2. ***1–Nearest Neighbor Classifier.*** In an attempt to be more sophisticated, let's suppose that for any new point $x \in \Omega$, we compute its *nearest neighbor* in the labelled data $\mathscr{X}_L$. Namely, we define the nearest neighbor function $N : \Omega \to \mathscr{X}_L$ via

$$N(x) := \operatorname*{argmin}_{x_i \in \mathscr{X}_L} |x - x_i|.$$

Note that if $x \in \mathscr{X}_L$, then $N(x) = x$. Now we define our nearest neighbor labelling function as

$$f(x) := y(N(x)).$$

In summary, to label a new point x, we find its nearest neighbor, query its label, and assign that label to x.

19.4.3. ***k–Nearest Neighbor Classifier.*** For a given integer $k \in \mathbb{N}$, we can define the k–Nearest Neighbor (k–NN) classifier to output labels by querying the labels of the k nearest points in the training data $\mathscr{X}_L$ to the point in question, allowing them to vote, and assigning x the label which obtained the most votes. For $k > 1$, we must specify a way of breaking ties. In particular, suppose we take $k = 2$, and for some point x, its two nearest neighbors have labels -1 and 1; how should we label x? The simplest manner is to flip a coin (or for $k > 2$ if there is more than a two-way tie, choose one of the tied labels uniformly at random). We could be more sophisticated, but for now let's assume this simple method of tie-breaking.

We will now discuss the basics of what are called Support Vector Machines, or SVMs for short. There are many ways to derive them, but I will give a geometric intuition for SVMs here. For a great introduction and reference text, I would recommend [14].

20.1. **Linear SVMs.** Our starting point will be the Geometric Hahn–Banach Theorem, whose statement here will seem fairly intuitive, but is of paramount importance to many applications both in pure and applied mathematics. To start, we need to define the notion of a convex set.

Definition 20.1. *A set $S \subset \mathbb{R}^n$ is **convex** if the line segment connecting every pair of points in S lies in S. That is, for every $x, y \in S$, the point $tx + (1-t)y$ is in S for every $t \in [0,1]$.*

Now we can state our seemingly simple theorem.

Theorem 20.2. *[Geometric Hahn–Banach Theorem] Let S_1 and S_2 be two nonempty closed, bounded (i.e., compact) convex sets in $\mathbb{R}^n$. If $S_1 \cap S_2 = \emptyset$, then S_1 and S_2 can be separated by a hyperplane. That is, there exists a nonzero $v \in \mathbb{R}^n$ and constant c such that*

$$\langle s_1, v\rangle > c > \langle s_2, v\rangle, \quad \text{for all } s_1 \in S_1,\ s_2 \in S_2.$$

A hyperplane is simply an $(n-1)$–dimensional subspace of $\mathbb{R}^n$ which is possibly translated elsewhere in the space. A natural question in Theorem 20.2 is: how can one find the hyperplane, or equivalently, the vector v? Suppose we could find $x_1 \in S_1$ and $x_2 \in S_2$ which minimized $|x-y|$ over all possible $x \in S_1$ and $y \in S_2$. Then we could simply draw the line segment connecting x_1 and x_2, and put the hyperplane passing through the midpoint of this line segment; that is, take $x = \frac{x_1+x_2}{2}$, and a hyperplane is defined by the point x and the normal vector being the vector connecting x to x_1.

Unfortunately, finding the points x_1 and x_2 is typically not possible, especially in a real data application when we have no idea what S_1 or S_2 really are. Typically, we will instead only have access to a discrete set of labelled points $\mathscr{X}_L = \{(x_i, y_i)\}_{i=1}^N$ where here we will let

$$y_i = \begin{cases} 1, & x_i \in S_1 \\ -1, & x_i \in S_2. \end{cases}$$

The choice of ± 1 labels is arbitrary, and evidently we could have chosen 1 and 2 instead, but it will make the rest of the analysis a bit easier. The natural question now is: if we assume S_1 and S_2 are as in Theorem 20.2, then given $\mathscr{X}_L$, how can we find the separating hyperplane?

To figure this out, let's first see how we would label a new point given a separating hyperplane and corresponding vector v and constant c as in Theorem 20.2. We can rearrange the inequality there to get

$$\langle x, v\rangle - c \begin{cases} > 0, & x \in S_1 \\ < 0, & x \in S_2. \end{cases} \tag{23}$$

Thus our decision, or labelling, function would simply be

$$f(x) := \operatorname{sgn}(\langle x, v\rangle - c) = \begin{cases} 1, & x \in S_1 \\ -1, & x \in S_2. \end{cases}$$

So, if we had v and c in hand, then our task would be done and we would have a perfect decision function.

At this point we should point out a theme that has been present but maybe unstated in the course thus far; probability and optimization are two of the most important tools at one's disposal in Data Science (on top of Linear Algebra of course, but you know that very well by now). In principal, we would want to find $c_{\mathscr{X}_L}$ and $v_{\mathscr{X}_L}$ which satisfied

$$v_{\mathscr{X}_L} = \underset{v\in\mathbb{R}^n, |v|=1}{\operatorname{argmax}} \operatorname{dist}(v, \mathscr{X}_L) \quad \text{subject to (23)},$$

where $\operatorname{dist}(v, \mathscr{X}_L) = \min_{x_i\in\mathscr{X}_L} |v - x_i|$. In other words, we want to find the unit vector which has the largest distance from the labelled data but still provides a separating hyperplane.

Note that we may multiply both $v_{\mathscr{X}_L}$ and $c_{\mathscr{X}_L}$ by a constant and the inequality (23) will be unchanged. Consequently, we could multiply by a constant α large enough that it guarantees that $\alpha\langle x, v\rangle - \alpha c \geq 1$ or ≤ 1 depending on if $x \in S_1$ or S_2. Notice that for the training data we may rewrite this as $y_i(\alpha\langle x, v\rangle - \alpha c) \geq 1$ because y_i is 1 if $x_i \in S_1$ and -1 if $x_i \in S_2$. Doing this allows us to replace the optimization function above by the following:

$$\min_{v\in\mathbb{R}^n, c\in\mathbb{R}} |v|^2 \quad \text{subject to} \quad y_i(\langle x_i, v\rangle - c) \geq 1, \quad i = 1, \ldots, N. \tag{24}$$

If v^*, c^* is the solution to (24), then we set $v_{\mathscr{X}_L} := \frac{v^*}{|v^*|}$ and $c_{\mathscr{X}_L} = \frac{c^*}{|v^*|}$, which gives us a unit normal vector describing the hyperplane.

This whole subsection has been about Linear SVMs; named so because the separation returned is a hyperplane, which is a linear subspace of $\mathbb{R}^n$ doing the separation of the data. However, it is evident that not all data can be linearly separated (consider our target data set discussed to motivate Spectral Clustering).

20.2. **The Kernel Trick.** We will shortly get to a discussion of Kernel SVMs, but to get there we will first discuss the so-called Kernel Trick. This has been used to great effect in Machine Learning overall. Figure 18 illustrates the basic idea.

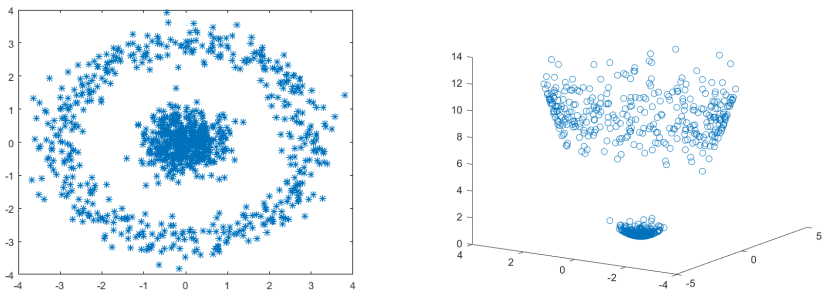

FIGURE 18. Target data set (left) which is clearly non-separable. A mapping of the data into $\mathbb{R}^3$ (right) which is now linearly separable by an SVM.

The Kernel Trick is in essence the idea that if we have data $\mathscr{X} \subset \mathbb{R}^n$ which is not linearly separable, we may be able to find a *feature map* $\phi : \mathbb{R}^n \to \mathbb{R}^D$ for some $D \gg n$ such that $\phi(\mathscr{X})$ is linearly separable in $\mathbb{R}^D$. So in stark contrast to dimensionality reduction, the Kernel Trick seeks to use dimensionality *inflation* in such a way as to allow for better

separation of data. Typically, the maps ϕ are highly nonlinear. As a note, the map ϕ yielding the result seen in Figure 18 is

$$\phi(x,y) := \begin{bmatrix} x \\ y \\ x^2+y^2 \end{bmatrix}.$$

So ϕ maps the points onto a positive cone in $\mathbb{R}^3$.

Some fancy higher level mathematics tells us actually that often, we don't actually need to explicitly compute the feature map ϕ. The reason for this is that kernels $k : \Omega \times \Omega \to \mathbb{R}$ which lead to positive semi-definite matrices $\{k(x_i,x_j)\}_{i,j=1}^N$ induce feature maps $\phi : \Omega \to \mathbb{R}^D$ which satisfy $k(x,y) = \langle \phi(x), \phi(y) \rangle$. That is, the kernel, which can be thought of as a notion of *similarity* between data (think back to the Gaussian similarity graph discussed surrounding Spectral Clustering), gives us inner product information of some associated feature map ϕ.

One has a choice to make here: one can either try to explicitly engineer a feature map ϕ and use this, or one can instead use a kernel which implicitly corresponds to some feature map. The benefit of using ϕ directly is that you have control over how to make it, but the drawback is that computing $\langle \phi(x_i), \phi(x_j) \rangle$ for the labelled data to estimate similarity can be hugely costly or completely impractical in many instances. The benefit of using a kernel k is that it typically involves low computational cost, but the drawback is that one has no specific control over the feature map ϕ. In fact, k is often chosen in an ad hoc sort of way rather than in a principled way. Let us stress that if one picks a kernel to use, one never actually forms or directly uses the feature map ϕ.

20.3. **Kernel SVMs.** The main idea of Kernel SVMs is that rather than use the training points $\mathscr{X}_L$, we use the dimension inflated version $\phi(\mathscr{X}_L) \subset \mathbb{R}^D$ implicitly through a kernel k; again, we never actually form or use ϕ in this case. Given a kernel k, the kernel SVM attempts to learn weights $\{w_i\}_{i=1}^N$ to make a labelling function $f : \mathbb{R}^n \to \{-1,1\}$ of the form

$$f(x) = \operatorname{sgn}\left(\sum_{i=1}^N w_i k(x,x_i) + c \right).$$

The constant c here represents potential bias of the data – again appealing to the Hahn–Banach example, this could be a shift that captures the middle of the line segment connecting the centroids of the convex sets, for example. Note that the Linear SVM case can be obtained by setting $k(x,y) := \langle x, y \rangle$. The question of how to learn the weights is resolved by solving a minimization problem based on the training data subject to a prescribed loss function.

Examples of common kernels are the Gaussian kernel, $k(x,y) = e^{-\frac{|x-y|^2}{\sigma^2}}$, polynomial kernels, $k(x,y) = \langle x, y \rangle^D$, and inverse multiquadric kernels, $k(x,y) = \frac{1}{\sqrt{1+|x-y|^2}}$. Note that Python's implementation of SVMs has 'rbf' as an option, which implements the Gaussian kernel for a given parameter that the user specifies.

Kernel SVMs generally find nonlinear, or curved decision boundaries, as opposed to Linear SVMs which always find hyperplanes. Thus, they are more generally applicable, as many data classes may not be linearly separable.

21. Neural Network Classifiers

At last we get to a discussion of Neural Networks! For our discussion, we will solely focus on NN Classifiers, which are nothing more than another method of deriving a labelling/classification function f given training data $\mathscr{X}_L$. So for us, NNs will fit squarely within the supervised learning framework we have been discussing lately.

21.1. **What is a Neuron?** A neuron in a NN provides the basic unit of computation.

REFERENCES

[1] Akram Aldroubi, Ali Sekmen, Ahmet Bugra Koku, and Ahmet Faruk Cakmak. Similarity matrix framework for data from union of subspaces. *Applied and Computational Harmonic Analysis*, 45(2):425–435, 2018.

[2] Francis J Anscombe. Graphs in statistical analysis. *The american statistician*, 27(1):17–21, 1973.

[3] David Arthur and Sergei Vassilvitskii. k-means++: The advantages of careful seeding. Technical report, Stanford, 2006.

[4] Afonso S Bandeira, Ben Blum-Smith, Joe Kileel, Amelia Perry, Jonathan Weed, and Alexander S Wein. Estimation under group actions: recovering orbits from invariants. *arXiv preprint arXiv:1712.10163*, 2017.

[5] Ronen Basri and David W Jacobs. Lambertian reflectance and linear subspaces. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (2):218–233, 2003.

[6] Andrea L Bertozzi and Arjuna Flenner. Diffuse interface models on graphs for classification of high dimensional data. *Multiscale Modeling & Simulation*, 10(3):1090–1118, 2012.

[7] Alan Kaylor Cline and Inderjit S Dhillon. Computation of the singular value decomposition. 2006.

[8] João Paulo Costeira and Takeo Kanade. A multibody factorization method for independently moving objects. *International Journal of Computer Vision*, 29(3):159–179, 1998.

[9] Edward W Forgy. Cluster analysis of multivariate data: efficiency versus interpretability of classifications. *biometrics*, 21:768–769, 1965.

[10] Keaton Hamm and Longxiu Huang. Perspectives on CUR decompositions. *Applied and Computational Harmonic Analysis*, In Press.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[12] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[13] Justin Matejka and George Fitzmaurice. Same stats, different graphs: generating datasets with varied appearance and identical statistics through simulated annealing. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 1290–1294, 2017.

[14] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[15] Gilbert Strang. *Linear algebra and learning from data*. Wellesley-Cambridge Press, 2019.

[16] René Vidal. Subspace clustering. *IEEE Signal Processing Magazine*, 28(2):52–68, 2011.

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

Appendix A. Formal Definitions from the Linear Algebra Review

Definition A.1. *A set V is a **vector space** over the real numbers if there are two operations (vector addition and scalar multiplication)*

$$+ : V \times V \to V, \qquad \times : \mathbb{R} \times V \to V$$

(although we write $x + y$ for $x, y \in V$ and αx for $\alpha \in \mathbb{R}$ and $x \in V$) such that the following properties are satisfied:

1. *(Additive Associativity)* $(x + y) + z = x + (y + z)$ *for all* $x, y, z \in V$
2. *(Additive Commutativity)* $x + y = y + x$ *for all* $x, y \in V$
3. *(Additive Identity) There exists an element* $0 \in V$ *such that* $x + 0 = x$ *for all* $x \in V$
4. *(Additive Inverse) For all* $x \in V$*, there exists* $-x \in V$ *such that* $x + (-x) = 0$
5. *(Compatibility of Scalar Multiplication)* $\alpha(\beta x) = (\alpha\beta)x$ *for all* $\alpha, \beta \in \mathbb{R}$ *and* $x \in V$
6. *(Multiplicative Identity)* $1x = x$ *for all* $x \in V$
7. *(Additive Distributivity)* $\alpha(x + y) = \alpha x + \alpha y$ *for all* $\alpha \in \mathbb{R}$ *and* $x \in V$
8. *(Multiplicative Distributivity)* $(\alpha + \beta)x = \alpha x + \beta x$ *for all* $\alpha, \beta \in \mathbb{R}$ *and* $x \in V$*.*

Definition A.2. *A vector space V over $\mathbb{R}$ is an **inner product space** provided it has an **inner product**, which is a two-variable function*

$$\langle \cdot, \cdot \rangle : V \times V \to \mathbb{R}$$

which satisfies the following properties:

1. *(Positive Definite)* $\langle x, x \rangle > 0$ *for all* $x \in V \setminus \{0\}$
2. *(Symmetry)* $\langle x, y \rangle = \langle y, x \rangle$
3. *(Linearity in the first argument)* $\langle \alpha x + \beta y, z \rangle = \alpha \langle x, z \rangle + \beta \langle y, x \rangle$ *for all* $\alpha, \beta \in \mathbb{R}$ *and* $x, y, z \in V$*.*

Note that inner products for vector spaces over the complex field $\mathbb{C}$ are slightly different as symmetry is replaced by conjugate symmetry ($\langle y, x \rangle = \overline{\langle x, y \rangle}$) and thus they are not linear in the second argument, but are conjugate linear.